

Лекция 1. Установка и запуск первой программы

Шаг 1. Установка компилятора

- Переход по ссылке: <https://golang.org/dl/>
- Установили для своей ОС компилятор GoLang
Важно : на данном курсе желательно, чтобы у вас была версия компилятора > 1.08

Шаг 2. GoRoot GoPath

Определение : GOROOT - это файловый путь, указывающий расположение **КОМПИЛЯТОРА** Go.

Определение: GOPATH - это файловый путь , указывающий на расположение **РАБОЧЕГО ОКРУЖЕНИЯ** (Там где пишем код и мазюкаем проекты). По умолчанию, на курсе мы создали GOPATH по адресу C:\Users\<username>\go

Шаг 3. Инициализация рабочего окружения

Чтобы создать рабочее окружение нам надо в GOPATH определить 3 директории:

- src - место, где будут лежать исходники проектов (скрипты .go)
- bin - место, где будут лежать скомпилированные бинарники, после выполнения компиляции проектов
- pkg - место, где будут жить сторонние пакеты для наших проектов

Шаг 4. Первая программа

В GOPATH/src создадим файл main.go со следующей начинкой:

```
package main

import "fmt"

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

Шаг 5. Запуск и компиляция

Go - компилируемый язык.

Для того, чтобы скомпилировать исполняемый файл, можно выполнить команду

```
go build <path/to/go/file.go>
```

Данная команда создает исполняемый файл по месту ее вызова. Это удобно, когда мы в пылу битвы и хочется посмотреть, скомпилируется ли оно вообще или в случае каких-либо тестов - позволяет на месте все проверять.

Другая команда, которая также позволяет создать исполняемый файл:

```
go install <path/to/go/file.go>
```

Данная команда создает исполняемый файл по пути GOPATH/bin.

Третья команда, которая будет часто использоваться на курсе - go run

```
go run <path/to/go/file.go>
```

Данная команда делает следующие действия:

- Создает исполняемый файл в временном хранилище
- Запускает этот файл
- И "устаряет его" (зависит ОС)

Для того, чтобы узнать, где это временное хранилище - -work

Шаг 6. Правильная структуризация рабочего окружения

Очень рекомендую создать следующий путь

GOPATH/src/github.com/<your_github_username>/<github_repo_name>

Лекция 2. Декларирование переменных и I/O

Шаг 1. Какая типизация?

В языке Go принята полустрогая статическая типизация.

Шаг 2. Способы декларирования переменных

Декларирование - это процесс связывания имени переменной с типом потенциального значения

При декларировании переменной автоматически происходит ее **инициализация НУЛЕВЫМ ЗНАЧЕНИЕМ ДЛЯ ЭТОГО ТИПА***.

Пример:

```
var age int
fmt.Println("My age is:", age)
```

Будет выведен 0.

Шаг 3. Декларирование и инициализация

Простейший случай единичной инициализации

```
//Декларирование и инициализация пользовательским значением
var height int = 183
fmt.Println("My height is:", height)

//В чем "полустрогость" типзации?
var uid = 12345
fmt.Println("My uid:", uid)
fmt.Printf("%T\n", uid)
```

Короткое присваивание

Оператор := слева от себя требует **КАК МИНИМУМ ОДНУ НОВУЮ ПЕРЕМЕННУЮ**.

```
//Множественное присваивание через :=
aArg, bArg := 10, 30
fmt.Println(aArg, bArg)
aArg, bArg = 30, 40
fmt.Println(aArg, bArg)
// aArg, bArg := 10, 30
// fmt.Println(aArg, bArg)

//Исключение из этого правила
bArg, cArg := 300, 400
fmt.Println(aArg, bArg, cArg)
```

```
package main

import (
    "fmt"
    "math"
)

func main() {
    //Простейший вывод на консоль. println - это вывод аргумента + '\n'
    fmt.Println("Hello world", "Hello another")
    fmt.Println("Second line")
    //Функция print - простой вывод аргумента
    fmt.Print("First")
    fmt.Print("Second")
    fmt.Print("Third")
    //Форматированный вывод: Printf - стандартный вывод os.Stdout с флагами
    форматирования
    fmt.Printf("\nHello, my name is %s\nMy age is %d\n", "Bob", 42)
    //////////////////////////////////////
    //////////////////////////////////////
    //Декларирование переменных
    var age int
    fmt.Println("My age is:", age)
    age = 32
    fmt.Println("Age after assignment:", age)
```

```

//Декларирование и инициализация пользовательским значением
var height int = 183
fmt.Println("My height is:", height)

//В чем "полустрогость" типзации? Можно опускать тип переменной
var uid = 12345
fmt.Println("My uid:", uid)
//Декларирование и инициализация переменных одного типа (множественный
случай)
var firstVar, secondVar = 20, 30
fmt.Printf("FirstVar:%d SecondVar:%d\n", firstVar, secondVar)
//Декларирование блока переменных
var (
    personName string = "Bob"
    personAge    = 42
    personUID int
)

fmt.Printf("Name: %s\nAge %d\nUID: %d\n", personName, personAge, personUID)

//Немного странного
var a, b = 30, "Vova"
fmt.Println(a, b)
a = 300
//Немного хорошего. Повторное декларирование переменной приводит к ошибке
компиляции
//var a = 200

//Короткая декларция (короткое объявление)
count := 10
fmt.Println("Count:", count)
count = count + 1
fmt.Println("Count:", count)
//Множественное присваивание через :=
aArg, bArg := 10, 30
fmt.Println(aArg, bArg)
aArg, bArg = 30, 40
fmt.Println(aArg, bArg)
// aArg, bArg := 10, 30
// fmt.Println(aArg, bArg)

//Исключение из этого правила
bArg, cArg := 300, 400
fmt.Println(aArg, bArg, cArg)

//Пример
width, length := 20.5, 30.2
fmt.Printf("Min dimensional of rectangle is : %.2f\n", math.Min(width,
length))
}

```

```

package main

import (
    "fmt"
    "os"
)

func main() {
    var (
        age  int
        name string
    )

    // fmt.Scan(&age)
    // fmt.Scan(&name)
    fmt.Scan(&age, &name)

    fmt.Printf("My name is: %s\nMy age is : %d\n", name, age)

    //Для ручного использования потока ввода
    fmt.Fscan(os.Stdin, &age)
    fmt.Println("New age:", age)
}

```

```

package main

import (
    "fmt"
    "strings"
    "unicode/utf8"
    "unsafe"
)

func main() {
    //Boolean => default false
    var firstBoolean bool
    fmt.Println(firstBoolean)
    //Boolean operands
    aBoolean, bBoolean := true, true
    fmt.Println("AND:", aBoolean && bBoolean)
    fmt.Println("OR:", aBoolean || bBoolean)
    fmt.Println("NOT:", !aBoolean)

    //Numerics. Integers
    //int8, int16, int32, int64, int
    //uint8, uint16, uint32, uint64, uint
    var a int = 32
    b := 92
    fmt.Println("Value of a:", a, "Value of b:", b, "Sum of a+b:", a+b)
    //Вывод типа через %T форматирование
    fmt.Printf("Type is %T\n", a)
    //Узнаем, сколько байт занимает переменная типа int

```

```

fmt.Printf("Type %T size of %d bytes\n", a, unsafe.Sizeof(a))

//Эксперимент. При использовании короткого объявления - тип для целого
числа - int платформо-зависимый
fmt.Printf("Type %T size of %d bytes\n", b, unsafe.Sizeof(b))

//Эксперимент 2. Используйте явное приведение типов при необходимости если
уверены что не произойдет коллизии
var first32 int32 = 12
var second64 int64 = 13
fmt.Println(int64(first32) + second64)

//Эксперимент 3. Если проводятся арифметические операции
// над int и intX , то обязательно нужно использовать механизм приведения.
Т.к. int != int64
var third64 int64 = 16123414
var fourthInt int = 156234
fmt.Println(third64 + int64(fourthInt))
// + - * / %

// Аналогичным образом устроены unit8, uint16, uint32, uint64, uint
//Numerics. Float
//float32, float64
floatFirst, floatSecond := 5.67, 12.54
fmt.Printf("type of a %T and type of %T\n", floatFirst, floatSecond)
sum := floatFirst + floatSecond
sub := floatFirst - floatSecond
fmt.Println("Sum:", sum, "Sub:", sub)
fmt.Printf("Sum: %.3f and Sub: %.3f\n", sum, sub)

//Numeric. Complex
c1 := complex(5, 7)
c2 := 12 + 32i
fmt.Println(c1 + c2)
fmt.Println(c1 * c2)

//Strings. Строка - это набор БАЙТ
name := "Федя"
lastname := "Pupkin"
concat := name + " " + lastname
fmt.Println("Full name:", concat)
fmt.Println("Length of string :", name, len(name)) // Функция len()
возвращает количество элементов в наборе
fmt.Println("Amount of chars:", name, utf8.RuneCountInString(name))
//Rune - руна. Это один utf-ный символ.
//Поиск подстроки в строке
totalString, subString := "ABCDEFGFG", "asd"
fmt.Println(strings.Contains(totalString, subString))
//rune -> alias int32
var sampleRune rune
var anotherRune rune = 'Q' // Для инициализации руны символьным значением -
используйте ''

```

```

var thirdRune rune = 234
fmt.Println(sampleRune)
fmt.Printf("Rune as char %c\n", sampleRune)
fmt.Printf("Rune as char %c\n", anotherRune)
fmt.Printf("Rune as char %c\n", thirdRune)
// "A" < "abcd"
fmt.Println(strings.Compare("abcd", "a")) // -1 if first < second, 0 if
first == second, 1 if first > second

var aByte byte // alias uint8
fmt.Println("Byte:", aByte)
}

```

```

package main

import (
    "fmt"
    "strings"
)

func main() {
    //Классический условный оператор
    // if condition {
    //     //body
    // }
    // }
    // Условный оператор с блоком else
    // if condition {

    // } else {

    // }

    var value int
    fmt.Scan(&value)

    if value%2 == 0 {
        fmt.Println("The number", value, "is even")
    } else {
        fmt.Println("The number", value, "is odd")
    }
    // if condition1 {

    // } else if condition2 {

    // } else if ... {

    // } else {

    // }

    var color string
    fmt.Scan(&color)

```

```

if strings.Compare(color, "green") == 0 {
    fmt.Println("Color is green")
} else if strings.Compare(color, "red") == 0 {
    fmt.Println("Color is red")
} else {
    fmt.Println("Unknown color")
}

//Good Инициализация в блоке условного оператора
//Блок присваивания - только :=
//Инициализируемая переменная видна ТОЛЬКО внутри области виидимости
условного оператора (в телах if, else if, или else)
// Но не за его пределами
if num := 10; num%2 == 0 {
    fmt.Println("EVEN")
} else {
    fmt.Println("ODD")
}

//Ущербно
/*
    var age int = 10
    if age > 7 {
        fmt.Println("Go to school")
    } //По факту, сюда подставляется ; компилятором, и дальнейший код уже
не имеет связи с предыдущим if
    else {
        fmt.Println("Another case")
    }
*/

//НЕ ИДЕОМАТИЧНО
if width := 100; width > 100 {
    fmt.Println("Width > 100")
} else {
    fmt.Println("Width <= 100")
}

//Странное правило номер 1: в Go стараются избегать блоков ELSE

//Идеоматичность
if height := 100; height > 100 {
    fmt.Println("height > 100")
    return
}
fmt.Println("Height <= 100")
}

```

```
package main
```

```
import (
    "fmt"
    "strings"

```


)

```
func main() {
    // for init; condition; post {
    // init - блок инициализации переменных цикла
    // condition - условие (если верно - то тело цикла выполняется, если нет -
то цикл завершается)
    // post - изменение переменной цикла (инкрементарное действие,
декрементарное действие)
    // }

    for i := 0; i <= 5; i++ {
        fmt.Printf("Current value: %d\n", i)
    }
    //Важный момент : в качестве init может быть использовано выражение
присваивания ТОЛЬКО через :=

    //break - команда, прерывающая текущее выполнение тела цикла и передающая
управление инструкциям, следующим
    // за циклом
    for i := 0; i <= 15; i++ {
        if i > 12 {
            break
        }
        fmt.Printf("Current value: %d\n", i)
    }
    fmt.Println("After for loop with BREAK")

    //continue - команда, прерывающая текущее выполнение тела цикла и
передающая управления СЛЕДУЮЩЕЙ ИТЕРАЦИИ ЦИКЛА
    for i := 0; i <= 20; i++ {
        if i > 10 && i <= 15 {
            continue
        }
        fmt.Printf("Current value: %d\n", i)
    }
    fmt.Println("After for loop with CONTINUE")

    //Вложенные циклы и лейблы
    for i := 0; i < 10; i++ {
        for j := 0; j <= i; j++ {
            fmt.Print("*")
        }
        fmt.Println()
    }
    fmt.Println("По идее выше треугольник")

    //Иногда бывает плохо. С лейблами по лучше. Лейблы - это синтаксический
сахар
outer:
    for i := 0; i <= 2; i++ {
        for j := 1; j <= 3; j++ {
```

```

        fmt.Printf("i:%d and j:%d and sum i+j=%d\n", i, j, i+j)
        if i == j {
            break outer // Хочу чтобы вообще все циклы (внешние тоже
остановились)
        }
    }
}
//Модификации цикла for.
//1. Классический цикл while do
var loopVar int = 0
// while (loopVar < 10){
// ....
// loopVar++
// }
for loopVar < 10 {
    fmt.Printf("In while like loop %d\n", loopVar)
    loopVar++
}
//2. Классический бесконечный цикл
var password string
outer2:
for {
    fmt.Print("Insert password: ")
    fmt.Scan(&password)
    if strings.Contains(password, "1234") {
        fmt.Println("Weak password . Try again")
    } else {
        fmt.Println("Password Accepted")
        break outer2
    }
}

//3. Цикл с множественными переменными цикла
for x, y := 0, 1; x <= 10 && y <= 12; x, y = x+1, y+2 {
    fmt.Printf("%d + %d = %d\n", x, y, x+y)
}
}

```

```
package main
```

```
import "fmt"
```

```

func main() {
    //Switch!
    var price int
    fmt.Scan(&price)
    //В switch-case запрещены дублирующие условия в case"ах
    switch price {
    case 100:
        fmt.Println("First case")
    case 110:
        fmt.Println("Second case")
    }
}

```

```

case 120:
    fmt.Println("Third case")
case 130:
    fmt.Println("Another case")
default:
    //0работает только в том случае, если не один из выше перечисленных
кейсов - не сработал
    fmt.Println("Default case")

}

//Case с множеством вариантов
var ageGroup string = "q" //Возрастные группы : "a", "b", "c", "d", "e"
switch ageGroup {
case "a", "b", "c":
    fmt.Println("AgeGroup 10-40")
case "d", "e":
    fmt.Println("AgeGroup 50-70")
default:
    fmt.Println("You are too yong/old")
}

// Case с выражениями
var age int
fmt.Scan(&age)

switch {
case age <= 18:
    fmt.Println("Too yong")
case age > 18 && age <= 30:
    fmt.Println("Second case")
case age > 30 && age <= 100:
    fmt.Println("Too old")
default:
    fmt.Println("Who are you")
}

//Case с проваливаниями. Проваливания выполняют ДАЖЕ ЛОЖНЫЕ КЕЙСЫ
//В момент выполнения fallthroug у следующего кейса не проверяется условие,
//а сразу выполняется тело
var number int
fmt.Scan(&number)
outer:
switch {
case number < 100:
    fmt.Printf("%d is less then 100\n", number)
    if number%2 == 0 {
        break outer
    }
    fallthrough
case number > 200:
    fmt.Printf("%d GREATER then 200\n", number)

```

```

        fallthrough
    case number > 1000:
        fmt.Printf("%d GREATER then 1000\n", number)
        fallthrough
    default:
        fmt.Printf("%d DEFAULT\n", number)
    }

    //Гадость с терминацией цикла for из switchv
    var i int
uberloop:
    for {
        fmt.Scan(&i)
        switch {
        case i%2 == 0:
            fmt.Printf("Value is %d and it's even\n", i)
            break uberloop
        }
    }

    fmt.Println("END")
}

```

```

package main

import "fmt"

func main() {
    //Массивы. Основа
    //1. Определение массива.
    //Создадим массив под хранение 5-ти целочисленных элементов
    var arr [5]int // При инициализации массива важно передать информацию -
    сколько элементов в нем будет
    fmt.Println("This is my array:", arr)
    //2. Определение элементов массива (после предварительной инициализации)
    // Необходимо обратиться к элементу массива через синтаксис arr[i] = elem
    arr[0] = 10
    arr[1] = 20
    arr[3] = -500
    arr[4] = 720
    fmt.Println("After elemtns init:", arr)
    //3. Определение массива с указанием элементов на месте
    // Если при инициализации количество элементов меньше номинальной длины
    массива
    // то недостающие элементы инициализируются нулями
    newArr := [5]int{10, 20, 30}
    fmt.Println("Short declaration and init:", newArr)
    //4. Создание массива через инициализацию переменных
    arrWithValues := [...]int{10, 20, 30, 40}
    fmt.Println("Arr declaration with [...]:", arrWithValues, "Length:",
len(arrWithValues))
    arrWithValues[0] = 10000
}

```

```

    fmt.Println("Arr declaration with [...]:", arrWithValues, "Length:",
len(arrWithValues))
    //5. Массив - это набор ЗНАЧЕНИЙ. То есть при всех манипуляциях - массив
копируется (жестко, на уровне компилятора)
    first := [...]int{1, 2, 3}
    second := first
    second[0] = 10000
    fmt.Println("First arr:", first)
    fmt.Println("Second arr:", second)
    //6. Массив и его размер - это две составляющие одного типа (Размер массив
- часть типа)
    // var aArr [5]int
    // var bArr [6]int
    // aArr[0] = 100
    // bArr = aArr
    // 7. Итерирование по массиву
    floatArr := [...]float64{12.5, 13.5, 15.2, 10.0, 12.0}
    for i := 0; i < len(floatArr); i++ {
        fmt.Printf("%d element of arr is %.2f\n", i, floatArr[i])
    }
    // 8. Итерирование по массиву через оператор range
    var sum float64
    for id, val := range floatArr {
        fmt.Printf("%d element of arr is %.2f\n", id, val)
        sum += val
    }
    fmt.Println("Total sum is:", sum)
    // 9. Игнорирование id в range based for цикле
    for _, val := range floatArr {
        fmt.Printf("%.2f value W0 id\n", val)
    }
    // 10. Многомерные массивы
    words := [2][2]string{
        {"Bob", "Alice"},
        {"Victor", "Jo"},
    }
    fmt.Println("Multidimensional array:", words)
    // 11. Итерирование по многомерному массиву
    for _, val1 := range words {
        for _, val2 := range val1 {
            fmt.Printf("%s ", val2)
        }
        fmt.Println()
    }
}

```

```
package main
```

```
import "fmt"
```

```
func main() {
```

```

//1. Слайсы (они же - срезы)
// Слайс - это динамическая обвязка над массивом.
startArr := [4]int{10, 20, 30, 40}
var startSlice []int = startArr[0:2] // Слайс инициализируется пустыми
квадратными скобками
fmt.Println("Slice[0:2]:", startSlice)
// Создали слайс, основываясь уже на существующем массиве

//2. Создание слайса без явной инициализации массива
secondSlice := []int{15, 20, 30, 40}
fmt.Println("SecondSlice:", secondSlice)

//3. Изменение элементов среза
originArr := [...]int{30, 40, 50, 60, 70, 80}
firstSlice := originArr[1:4] // Это набор ссылок на элементы нижележащего
массива
for i, _ := range firstSlice {
    firstSlice[i]++
}
fmt.Println("OriginArr:", originArr)
fmt.Println("FirstSlice:", firstSlice)

//4. Один массив и два производных среза
fSlice := originArr[:]
sSlice := originArr[2:5]

fmt.Println("Before modifications: Arr:", originArr, "fSlice:", fSlice,
"sSlice:", sSlice)
fSlice[3]++
sSlice[1]++
fmt.Println("After modifications: Arr:", originArr, "fSlice:", fSlice,
"sSlice:", sSlice)

//5. Срез как встроенный тип
// type slice struct {
//     Length int
//     Capacity int
//     ZeroElement *byte
// }

//6. Длина и емкость слайса
wordsSilce := []string{"one", "two", "three"}
fmt.Println("slice:", wordsSilce, "Length:", len(wordsSilce), "Capacity:",
cap(wordsSilce))
wordsSilce = append(wordsSilce, "four")
fmt.Println("slice:", wordsSilce, "Length:", len(wordsSilce), "Capacity:",
cap(wordsSilce))
// Capacity (cap) или ёмкость слайса - это значение, показывающее СКОЛЬКО
ЭЛЕМЕНТОВ В ПРИНЦИПЕ
// можно добавить в срез БЕЗ ВЫДЕЛЕНИЯ ДОПОЛНИТЕЛЬНОЙ ПАМЯТИ ПОД
НИЖЕЛЕЖАЩИЙ МАССИВ.

```

```

    // Допустим у нас есть срез на 3 элемента (инициализировали без явного
указания массива)
    // Компилятор при создании этого среза СНАЧАЛА создал массив ровно на 3
элемента
    // После этого компилятор вернул адрес, где этот массив живет
    // Срез запомнил этот адрес и теперь ссылается на него
    // Потом
    // Начинаем деформировать слайс (увеличим длину /увеличим количество
элементов)
    // Проблема - в нижележащем массиве (на котором основан слайс) все 3
ячейки. Что делать?
    // Компилятор ищет в памяти место для массива размера 3*2 (в общем случае
n*2, где n - первоначальный размер)
    // После того как место найдено (в нашем случае найдено место для 6
элементов), в это место копируются
    // старые 3 элемента на свои позиции. На 4-ую позицию мы добавляем новый
элемент
    // После этого компилятор возвращает нашему слайсу новый адрес в памяти,
где находится массив под 6 элементов.

    //Емкость всегда будет изменять как n*2.
    numerics := []int{1, 2}
    for i := 0; i < 200; i++ {
        if i%5 == 0 {
            fmt.Println("Current len:", len(numerics), "Current cap:",
cap(numerics))
        }
        numerics = append(numerics, i)
    }

    //Важно: после выделения памяти под новый массив, ссылки со старым будут
перенесены в новый
    // Пример
    numArr := [2]int{1, 2}
    numSlice := numArr[:]

    numSlice = append(numSlice, 3) // В этот момент numSlice больше не
ссылается на numArr
    numSlice[0] = 10
    fmt.Println(numArr)
    fmt.Println(numSlice)

    // 7. Как создавать слайсы наиболее эффективно?
    // make() - это функция, позволяющая более детально создавать срезы
    sl := make([]int, 10, 15)
    // []int - тип коллекции
    // 10 - длина
    // 15 - емкость
    //Сначала инициализируется arr = [15]int
    //Затем по нему делается срез arr[0:10]
    //После чего он возвращается
    fmt.Println(sl)

```

```

// 8. Добавление элементов в CPE3
myWords := []string{"one", "two", "three"}
fmt.Println("myWords:", myWords)
anotherSlice := []string{"four", "five", "six"}
myWords = append(myWords, anotherSlice...)
myWords = append(myWords, "seven", "eight")
fmt.Println("myWords:", myWords)
//9. Многомерный срез
mSlice := [][]int{
    {1, 2},
    {3, 4, 5, 6},
    {10, 20, 30},
    {},
}
fmt.Println(mSlice)
}

```

```

package main

import (
    "fmt"
    "unicode/utf8"
)

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

    //1. Строка - это байтовый слайс со своими особенностями при обращении
    //к нижележащему массиву
    word := "Тестовая строка"
    fmt.Printf("String %s\n", word)
    // Какие значения байт сейчас находятся в слайсе word?
    fmt.Printf("Bytes: ")
    for i := 0; i < len(word); i++ {
        fmt.Printf("%x ", word[i]) // %x - формат представления 16-ти ричного
байта
    }
    fmt.Println()
    // Каким образом получать доступ к отдельно стоящим символам?
    fmt.Printf("Characters: ")
    for i := 0; i < len(word); i++ {
        fmt.Printf("%c ", word[i]) // %c - формат представления символа
    }
    fmt.Println()
    //2. Строки в Go - хранятся как наборы UTF-8символов. Каждый символ, вообще
говоря, может занимать
    // больше чем 1 байт

    //3. Руна (Rune) - это стандартный встроенный тип в Go (alias над int32),
позволяющий хранить

```



```

    //единый неделимый UTF символ ВНЕ ЗАВИСИМОСТИ ОТ ТОГО сколько байт он
занимает
    fmt.Printf("Runes: ")
    runeSlice := []rune(word) // Преобразование слайса байт к слайсу рун
[]byte(sliceRune)
    for i := 0; i < len(runeSlice); i++ {
        fmt.Printf("%c ", runeSlice[i])
    }
    fmt.Println()
    //4. Итерирование по строке с использованием рун
    for id, runeVal := range word { // id - это индекс байта, с КОТОРОГО
НАЧИНАЕТСЯ РУНА runeVal
        fmt.Printf("%c starts at postion %d\n", runeVal, id)
    }

    //5. Создание строки из слайса байт
    myByteSlice := []byte{0x40, 0x41, 0x42, 0x43} // Исходное представление
байтов
    myStr := string(myByteSlice)
    fmt.Println(myStr)

    myDecimalByteSlice := []byte{100, 101, 102, 103} // Синтаксический сахар -
можно использовать десятичное представление байтов
    myDecimalStr := string(myDecimalByteSlice)
    fmt.Println(myDecimalStr)

    //6. Создание строки из слайса рун
    // Руны как hex
    runeHexSlice := []rune{0x45, 0x46, 0x47, 0x48}
    myStrFromRune := string(runeHexSlice)
    fmt.Println("From Runes(hex):", myStrFromRune)
    // Руны как литералы
    runeLiteralSlice := []rune{'V', 'a', 's', 'y', 'a'} // '' - таким образом
обозначается руна
    myStrFromRuneLiterals := string(runeLiteralSlice)
    fmt.Println("From Runes(literals):", myStrFromRuneLiterals)

    fmt.Printf("%s and %T\n", myStrFromRuneLiterals, myStrFromRuneLiterals)

    //7. Длина и емкость строки
    // Длина len() - количество байт в слайсе
    fmt.Println("Length of Вася:", len("Вася"), "bytes")
    // Длина RuneCounter - количество !рун!
    fmt.Println("Length of Вася:", utf8.RuneCountInString("Вася"), "runes")
    // Вычисление емкости строки - бессмысленно, т.к. строка базовый тип
    fmt.Println(cap([]rune("Вася")))

    //8. Сравнение строк == и !=. Начиная с go 1.6
    word1, word2 := "Вася", "Петя"
    if word1 == word2 {
        fmt.Println("Equal")
    } else {

```

```
        fmt.Println("Not equal")
    }

    //9. Конкатенация
    word3 := word1 + word2
    fmt.Println(word3)

    //10. Строитель строк (java -> StringBuiler)
    firstName := "Alex"
    secondName := "Johnson"
    fullName := fmt.Sprintf("%s #### %s", firstName, secondName)
    fmt.Println("FullName:", fullName)

    //11. Строки не изменяемые
    // fullName[0] = "Q"

    //12. А слайсы изменяемые :)
    fullNameSlice := []rune(fullName)
    fullNameSlice[0] = 'F'
    fullName = string(fullNameSlice)
    fmt.Println("String mutation:", fullName)

    //13. Сравнение рун
    if 'Q' == 'Q' {
        fmt.Println("Runes equal")
    } else {
        fmt.Println("Runes not equal")
    }

    //14. Где живут полезные методы работы со строками?
    // import "strings"
}
```

```

package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
)

func main() {
    var name string
    input := bufio.NewScanner(os.Stdin)
    if input.Scan() { // Команда захвата потока ввода и сохранения в буфер
        (захват идет до символа окончания строки)
        name = input.Text() // Команда возвращения элементов, помещенных в
        буфер (отдаст string)
    }
    fmt.Println(name)

    fmt.Println("For loop started:")
    for {
        if input.Scan() {
            result := input.Text()
            if result == "" {
                break
            }
            fmt.Println("Input string is:", result)
        }
    }

    //Преобразование строкового литерала к чему-нибудь числовому
    numStr := "10"
    numInt, _ := strconv.Atoi(numStr) // Atoi - Anything to Int (именно int)
    fmt.Printf("%d is %T\n", numInt, numInt)

    numInt64, _ := strconv.ParseInt(numStr, 10, 64)
    numFloat32, _ := strconv.ParseFloat(numStr, 32) // Но это 64-разрядное
    число будет без проблем ГАРАНТИРОВАННО ПРЕВОДИТЬСЯ К 32
    fmt.Println(numInt64, numFloat32)
    fmt.Printf("%.3f and %T\n", numFloat32, float32(numFloat32))
}

```

```

package main

import "fmt"

func main() {
    //1. Map - это набор пар ключ:значение. Инициализация пустой карты
    mapper := make(map[string]int)
    fmt.Println("Empty map:", mapper)
}

```

```

//2. Добавление пар в существующую карту
mapper["Alice"] = 24
mapper["Bob"] = 25
fmt.Println("Mapper after adding pairs:", mapper)

//3. Инициализация карты с указанием пар
newMapper := map[string]int{
    "Alice": 1000,
    "Bob": 1000,
}
newMapper["Jo"] = 3000
fmt.Println("New Mapper:", newMapper)

//4. Что может быть ключом в карте?
//4.1 ВАЖНО: Карта НЕ УПОРЯДОЧЕНА В GO
//4.2 КЛЮЧОМ В КАРТЕ МОЖЕТ БЫТЬ ТОЛЬКО СРАВНИМЫЙ ТИП (==, !=)

//5. Нулевое значение для карты
// var mapZeroValue map[string]int // mapZeroValue == nil
// mapZeroValue["Alice"] = 12

//6. Получение элементов из карты
//6.1 Получение элемента, который представлен в карте
testPerson := "Alice"
fmt.Println("Salary of testPerson:", newMapper[testPerson])
//6.2 Получение элемента, который НЕ представлен в карте
testPerson = "Derek"
fmt.Println("Salary of new testPerson:", newMapper[testPerson]) // При
обращении по несуществующему ключу - новая пара не добавляется
fmt.Println(newMapper)

//7. Проверка вхождения ключа
employee := map[string]int{
    "Den": 0,
    "Alice": 0,
    "Bob": 0,
}

//7.1 При обращении по ключу - возвращается 2 значения
if value, ok := employee["Den"]; ok {
    fmt.Println("Den and value:", value)
} else {
    fmt.Println("Den does not exists in map")
}

if value, ok := employee["Jo"]; ok {
    fmt.Println("Jo and value:", value)
} else {
    fmt.Println("Jo does not exists in map")
}

//8. Перебор элементов карты

```

```

fmt.Println("=====")
for key, value := range employee {
    fmt.Printf("%s and value %d\n", key, value)
}

//9. Как удалять пары
//9.1 Удаление существующей пары
fmt.Println("Before deleting:", employee)
delete(employee, "Den")
fmt.Println("After first deleting:", employee)

//9.2 Удаление не существующей пары
if _, ok := employee["Anna"]; ok {
    delete(employee, "Anna") // ОЧЕНЬ ДОРОГАЯ ОПЕРАЦИЯ
}

fmt.Println("After second deleting:", employee)

//10. Количество пар == длина карты
fmt.Println("Pair amount in map:", len(employee))

//11. Карта (как и слайс) ссылочный тип
words := map[string]string{
    "One": "Один",
    "Two": "Два",
}

newWords := words
newWords["Three"] = "Три"
delete(newWords, "One")
fmt.Println("words map:", words)
fmt.Println("newWords map:", newWords)

//12 . Сравнение карт
//12.1 Сравнение массивов (массив можно использовать как ключ в карте)
if [3]int{1, 2, 3} == [3]int{1, 2, 3} {
    fmt.Println("Equal")
} else {
    fmt.Println("Not equal")
}

//12.2 Сравнение слайсов. Слайсы (из-за того что тип ссылочный - можно
сравнить на равенство только с nil)
// if []int{1, 2, 3} == []int{1, 2, 3} {
//     fmt.Println()
// }

//12.3 Сравнение карт. Карта (из-за того, что тип ссылочный - можно
сравнивать только с nil)
aMap := map[string]int{
    "a": 1,
}
var bMap map[string]int

```

```
    if aMap == nil {
        fmt.Println("Zero value map")
    }

    if bMap == nil {
        fmt.Println("Zero value of map bMap")
    }

    //13. Грустное следствие
    // Если мапа/слайс являются составляющими какой-либо структуры - структура
    автоматически не сравнима
}
```

<!-- 05 Lec11 -->

<!-- markdown-pdf -o summary.pdf summary.md -->