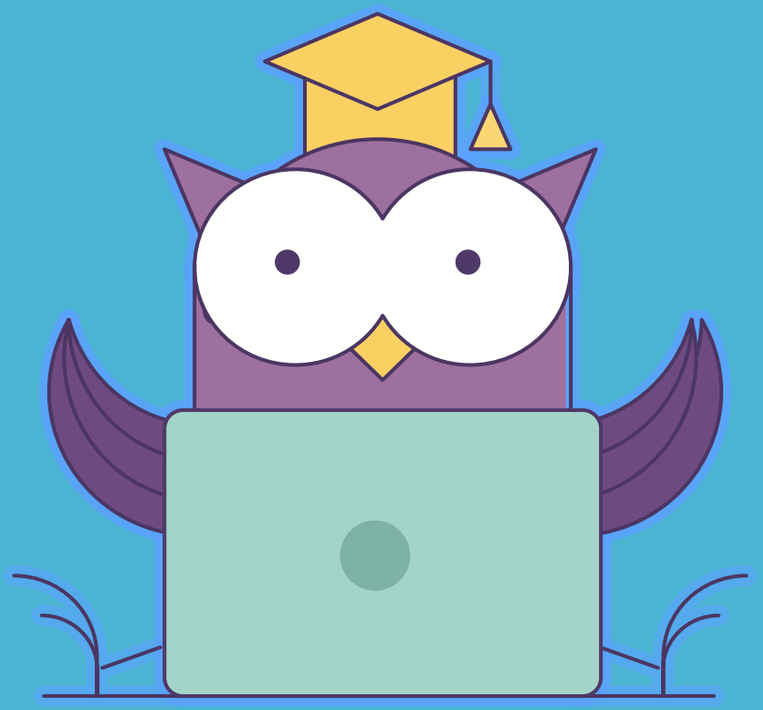
ОНЛАЙН-ОБРАЗОВАНИЕ

Элементарные типы данных в Go

Дмитрий Смаль

Как меня слышно и видно?



> Напишите в чат

+ если все хорошо

- если есть проблемы со звуком или с видео

!включить запись!

У программ есть несколько стандартных файловых дескрипторов

- STDIN (`os.Stdin`)
- STDOUT (`os.Stdout`)
- STDERR (`os.Stderr`)

Сообщения для пользователя нужно выводить в `os.Stdout`, например через `fmt.Printf`.
Ошибки нужно выводить в `os.Stderr`, например через `log.Printf`.

Если ошибка фатальная, нужно прервать выполнение программы и вернуть *ненулевой* код выхода, например с помощью `os.Exit(1)` или `log.Fatalf("message")` или `panic`

Небольшой тест

Пожалуйста, пройдите небольшой тест.

Возможно вы уже многое знаете про типы данных в Go =)

https://forms.gle/zHXnyDAkTLyyQaAK8

Объявление переменных в Go

```go
var Storage map[string]string          // zero value

var storage = make(map[string]string) // автовывод типа

func Answer() int {
  return 42
}

func main() {
  var i int = 10
  j := i  // короткое объявление, только внутри функций
}
```

Публичные и приватные идентификаторы

Публичные идентификаторы - те, которые видны за пределами вашего *пакета*. Публичные идентификаторы начинаются с заглавной буквы `Storage`, `Printf`.

Приватные идентификаторы - начинаются со строчной буквы `i`, `j` и видны только в вашем пакете. Структуры могут содержать как приватные так и публичные поля.

```
type User struct {
  Name     string  // будет видно в json.Marshal
  password string  // не будет видно
}
```

Какие есть элементарные типы данных в Go ?

- Целые: `int`, `uint`, `int8`, `uint8`, ...
- Алиасы к целым: `byte` = `uint8`, `rune` = `int32`
- С плавающей точкой: `float32`, `float64`
- Комплексные: `complex64`, `complex128`
- Строки: `string`
- Указатели: `uintptr`, `*int`, `*string`, ...

- Есть значение "по умолчанию" - это `0`
- Типы `int` и `uint` могут занимать 32 и 64 бита на разных платформах
- Нет автоматического преобразования типов
- `uintptr` - целое число, не указатель

Преобразование типов

В Go всегда необходимо *явное преобразование* типов

```
var i int32 = 42
var j uint32 = i          // ошибка
var k uint32 = uint32(i) // верно
var n int64 = i           // ошибка!
var m int64 = int64(i)    // верно
var r rune = i            // верно ?
```

За редким исключением: https://golang.org/ref/spec#Properties_of_types_and_values

Литералы числовых типов

Все довольно стандартно

```
42          // десятичная система
0755        // восьмеричная система
0xDeadBeaf // шестнадцатеричная, hex

3.14        // с плавающей точкой
.288
2.e+10

1+1i        // комплексные
```

Операции над числами

Все так же стандартно

```
+    sum                    integers, floats, complex values, strings
-    difference             integers, floats, complex values
*    product                integers, floats, complex values
/    quotient               integers, floats, complex values
%    remainder              integers

&    bitwise AND            integers
|    bitwise OR             integers
^    bitwise XOR            integers
&^   bit clear (AND NOT)    integers

<<   left shift             integer << unsigned integer
>>   right shift            integer >> unsigned integer
```

Строки в Go - это *неизменяемая* последовательность байтов (`byte` = `uint8`)

```
// src/runtime/string.go
type stringStruct struct {
    str unsafe.Pointer
    len int
}
```

Хорошо описано тут: https://blog.golang.org/strings

Что можно делать со строками ?

```
s := "hello world"        // создавать
var c byte = s[0]        // получать доступ к байту(!) в строке
var s2 string = s[5:10]  // получать подстроку (в байтах!)
s2 := s + " again"        // склеивать
l := len(s)               // узнавать длину в байтах
```

Задачка

Написать функцию `itoa` (integer to ascii), которая принимает на вход целое число и возвращает строку с этим же числом

https://play.golang.org/p/K54lV4LnvzV

Исходники Go программ и все литералы - в кодировке `UTF-8`
Как устроен UTF-8 ?

- `Z` = `5A`
- `Я` = `D0` `AF`
- 🎬 = `E2` `99` `AC`

Количество символов в строке != длинна строки
`s[i]` *- это i-ый байт, не символ*

Символы Unicode в Go представлены с помошью типа `rune` = `int32`
Литералы рун выглядят так

```
var r rune = 'Я'
var r rune = '\n'
var r rune = '本'
var r rune = '\xff'   // последовательность байт
var r rune = '\u12e4' // unicode code-point
```

Для удобной работы с Unicode и UTF-8 используем пакет `unicode` и `unicode/utf8`

```
// получить первую руну из строки и ее размер в байтах
DecodeRuneInString(s string) (r rune, size int)

// получить длинну строки в рунах
RuneCountInString(s string) (n int)

// проверить валидность строки
ValidString(s string) bool
```

Преобразование в слайс

Вы всегда можете преобразовать строку в слайс байтов или рун и работать далее со слайсом

```
s := "привет"
ba := []byte(s)
ra := []rune(s)
fmt.Printf("% v\b\n", ba)
fmt.Printf("% v\n\n", ra)
```

https://play.golang.org/p/hCSF7LWU24B

Итерация по строке

OTUS

По байтам

```
for i := 0; i < len(s); i++ {
    b := s[i]
    // i строго последоваельно
    // b имеет тип byte, uint8
}
```

По рунам

```
for i, r := range s {
    // i может перепрыгивать значения 1,2,4,6,9...
    // r - имеет тип rune, int32
}
```

Стандартная библиотека

В Go есть обширная библиотека для работы со строками - пакет `strings`

```
// проверка наличия подстроки
Contains(s, substr string) bool

// строка начинается с ?
HasPrefix(s, prefix string) bool

// склейка строк
Join(a []string, sep string) string

// разбиение по разделителю
Split(s, sep string) []string
```

Эффективная склейка строк

OTUS

Т.к. строки read-only, каждая склейка через `+` или `+=` приводит к выделению памяти. Что бы оптимизировать число аллокаций используйте `strings.Builder`

```
import "strings"

var b strings.Builder
for i := 33; i >= 1; i-- {
    b.WriteString("Код")
    b.WriteRune('ъ')
}
result := b.String()
```

Константы

Константы - неизменяемые значения, доступные только во время компиляции.

```
const PI = 3              // принимает подходящий тип
const pi float32 = 3.14   // строгий тип
const (
  TheA = 1
  TheB = 2
)
const (
  X = iota    // 0
  Y           // 1
  Z           // 2
)
```

Во время компиляции ?

```
package main
const HelloConst = 3
var HelloVar = 5
func main() {
    print(HelloVar, HelloConst)
}
```

Скомпилируем и посмотрим символы

```
$ go build -o 1.out 1.go
$ go tool nm 1.out  | grep Hello
 10bd148 D main.HelloVar
```

Домашнее задание

Создать Go функцию, осуществляющую примитивную распаковку строки, содержащую повторяющиеся символы / руны.
Примеры:

- `"a4bc2d5e"` => `"aaaabccddddde"`
- `"abcd"` => `"abcd"`
- `"45"` => `""` (некорректная строка)
- `"qwe\4\5"` => `"qwe45"` (*)
- `"qwe\45"` => `"qwe44444"` (*)
- `"qwe\\5"` => `"qwe\\\\\"` (*)

Небольшой тест

Проверим что мы узнали за этот урок

https://forms.gle/zHXnyDAkTLyyQaAK8

Заполните пожалуйста опрос

https://otus.ru/polls/3567/

Спасибо за внимание!