Backend Golang
Урок 3

Типы данных. Функции.
Управление потоком выполнения. Ошибки

Цели занятия

- Научиться работать с элементарными типами данных
- Научиться работать с функциями
- Научиться управлять потоком выполнения с помощью for, if-else, switch, defer

Объявление переменных

```
package main

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 идентификаторы (экспортируемые)

```go
package main

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
}
```

Приватные идентификаторы

```
package main

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
}
```

Поля структур

```
type Person struct {
   Name string // public
   age  int    // private
}
```

Элементарные типы данных

- Целые числа: int, uint, int32, int64, etc…
- Числа с плавающей точкой: float32, float64
- Комплексные: complex64, complex128
- Строка: string
- Логический: bool
- Указатели: uintptr, *int, *string, **int, ******int ...

Псевдонимы типов (type alias)

```
// byte is an alias for uint8 and is equivalent to uint8 in all ways. It is
// used, by convention, to distinguish byte values from 8-bit unsigned
// integer values.
type byte = uint8

// rune is an alias for int32 and is equivalent to int32 in all ways. It is
// used, by convention, to distinguish character values from integer values.
type rune = int32
```

Значения по умолчанию (нулевые значения)

```
var (
  a       int     // 0
  b       float32 // 0
  isLocal bool    // false
  intPtr  *int    // nil
  name    string  // ""
)
```

Явное преобразование типов

```
var (
  a int64 = 8
  b int   = a // ошибка

  c int = int(a) // верно
)
```

Строки

Строки — неизменяемая последовательность байтов

ну если сильно не извращаться: https://play.golang.org/p/LpuiZcNzA4x

```
alikhan@cerebro:~$ grep -A 4 "type stringStruct" -w /usr/local/go/src/runtime/string.go
type stringStruct struct {
        str unsafe.Pointer
        len int
}
```

Операции над строками

```
s := "инициализация из строкового литерала"
var c byte = s[0]      // получение байта по индексу
var s2 string = s[2:5] // срез строки
s2 += " склеивание строк"
l := len(s2) // длина строки
```

Задание 1

Написать функцию reverse, которая принимает строки и возвращает строку с символами в обратном порядке

https://play.golang.org/p/IjOW5LV0YMW

```
func reverse(s string) string {
  // TODO: implement
  return ""
}
```

Задание 2

Написать функцию `itoa` (integer to ascii), которая принимает на вход `int` и возвращает это же число в виде строки

https://play.golang.org/p/p7vv5kp0eZi

```
func itoa(i int) (s string) {
  // TODO: implement
  return s
}
```

Unicode

Кодировка исходного кода go-программы — в кодировке UTF-8.

Символ может занимать от 1 до 4 байт (8 - 32 бит)

Руны — это символы

type rune = int32

```
fmt.Println(utf8.RuneLen('A')) // 1
fmt.Println(utf8.RuneLen('Ы')) // 2
fmt.Println(utf8.RuneLen('♫')) // 3
```

Количество символов в строке

```
s := "abc абв"
fmt.Println(len(s))                    // 10
fmt.Println(utf8.RuneCountInString(s)) // 7
```

Неправильное получение символов 1

```
package main

import "fmt"

func main() {
  s := "abc абв"
  for i := 0; i < len(s); i++ {
     fmt.Print(string(s[i]))
  }
  // abc_Ð°Ð±Ð²
}
```

Неправильное получение символов 2

```
package main

import "fmt"

func main() {
  s := "abc абв"
  for i := range s {
     fmt.Print(string(s[i]))
  }
  // abc_ÐÐÐ
}
```

Правильная итерация по символам

```go
package main

import "fmt"

func main() {
  s := "abc абв"
  for i, r := range s {
     fmt.Printf("#%d: %c\n", i, r)
  }
  // #0: a
  // #1: b
  // #2: c
  // #3:
  // #4: a
  // #6: б
  // #8: в
}
```

```go
package main

import "fmt"

func main() {
  s := "abc абв"
  bytes := []byte(s)
  for i, b := range bytes {
     fmt.Printf("#%d: %c\n", i, b)
  }
  // #0: a
  // #1: b
  // #2: c
  // #3:
  // #4: Đ
  // #5: °
  // #6: Đ
  // #7: ±
  // #8: Đ
  // #9: ²
}
```

Преобразование в слайс байтов (массив на стероидах)

```go
package main

import "fmt"

func main() {
  s := "abc абв"
  bytes := []rune(s)
  for i, b := range bytes {
     fmt.Printf("#%d: %c\n", i, b)
  }
  // #0: a
  // #1: b
  // #2: c
  // #3:
  // #4: а
  // #5: б
  // #6: в
}
```

Преобразование в слайс рун (символов)

Неэффективная склейка строк

```
func Concatenation() {
  var s string
  for i := 0; i < 1000; i++ {
     s += strconv.Itoa(i)
  }
}
```

Эффективная склейка строк

```
func Builder() {
  var b strings.Builder
  for i := 0; i < 1000; i++ {
     b.WriteString(strconv.Itoa(i))
  }
}
```

Сравнение

```
package main

import "testing"

func BenchmarkConcatenation(b *testing.B) {
  for i := 0; i < b.N; i++ {
     Concatenation()
  }
}

func BenchmarkBuilder(b *testing.B) {
  for i := 0; i < b.N; i++ {
     Builder()
  }
}
```

Benchmark

```
alikhan@cerebro:~/go/src/github.com/alikhanmurzayev/test_project$ go test -bench=.
goos: linux
goarch: amd64
pkg: github.com/alikhanmurzayev/test_project
cpu: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
BenchmarkConcatenation-8            4504            293467 ns/op
BenchmarkBuilder-8                 34598             29934 ns/op
PASS
ok      github.com/alikhanmurzayev/test_project 3.672s
alikhan@cerebro:~/go/src/github.com/alikhanmurzayev/test_project$
```

Со строками пока всё, переходим к функциями

Области видимости и блоки

Блок (block) — это, возможно пустая, последовательность объявлений (declarations) и операторов (statements) в объёмлющих фигурных скобках.

```
func main() {
  if len(os.Args) > 1 {
     fmt.Println(os.Args[1:])
  }
}
```

Объявления

Объявление (declaration) связывает непустой идентификатор с константой, типом, переменной, функцией, меткой (label) или пакетом. Каждый идентификатор в программе должен быть объявлен. Ни один идентификатор не может быть объявлен дважды в одном и том же блоке, и ни один идентификатор не может быть объявлен как в блоке файла, так и в блоке пакета.

```
var a int

type Student struct {
  name string
}

const Pi float64 = 3.14159265358979323846
```

Виды блоков

- universe block — весь код проекта
- package block — весь код пакета
- file block — исходный код в файле
- local block — {}

Неявные блоки: for, if, switch, case, select

```
// {
for i := 0; i < 5; i++ {
  fmt.Println(i)
}
// }
```

```
if i := 0; i >= 0 {
  fmt.Println(i)
}
```

```
switch i := 2; i * 4 {
case 8:
  j := 0
  fmt.Println(i, j)
default:
  // "j" is undefined here
  fmt.Println("default")
} // "j" is undefined here
```

Область видимости

```
package main

func main() {
   {
      {
         var a = 22
         println(a)
      }
      println(a) // unresolved reference
   }
}
```

Shadowing

```
package main

func main() {
   {
      var a = 4
      println(a) // 4
      {
         println(a) // 4
         var a = 22
         println(a) // 22
      }
   }
}
```

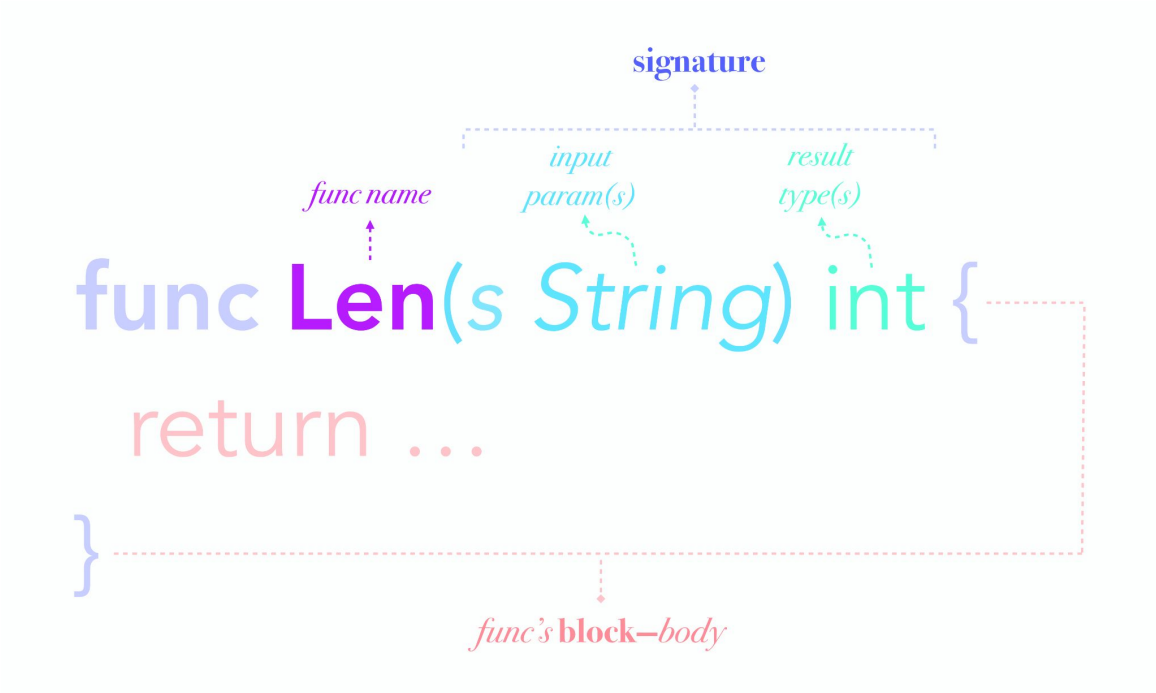
Сколько раз мы объявили x?

```
package main

import "fmt"

func f(x int) {
  for x := 0; x < 10; x++ {
     fmt.Println(x)
  }
}

var x int

func main() {
  var x = 200
  f(x)
}
```

```go
package main

import (
  "fmt"
  "strconv"
)

func parseInt(s string) (int, error) {
  n, err := strconv.Atoi(s)
  if err != nil {
     b, err := strconv.ParseBool(s)
     if err != nil {
        return 0, err
     }
     if b {
        n = 1
     }
  }
  return n, err
}

func main() {
  fmt.Println(parseInt("true"))
}
```

Какая ошибка
будет возвращена?

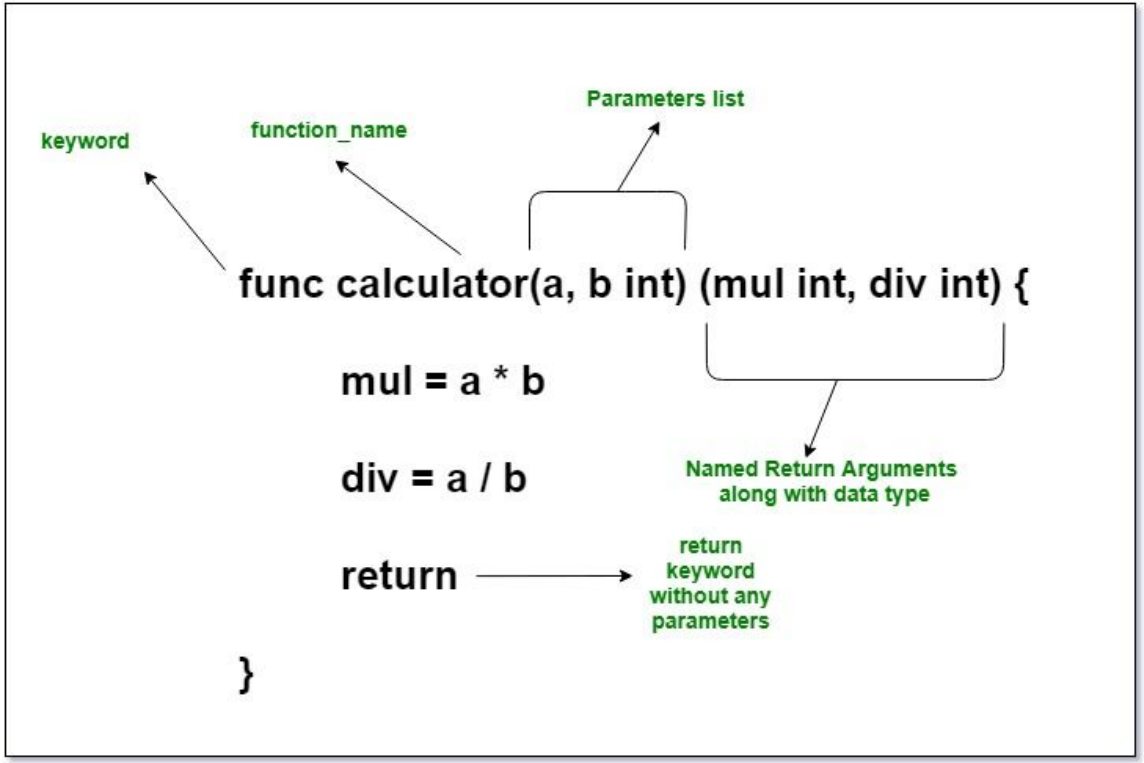
Объявление функции



Особенности функций

- Нет значений по умолчанию
- Функция может возвращать несколько значений
- Функция — объект первого класса, можем работать как с обычным значением
- параметры в функцию передаются по значению

Еще пример функции



Именованные функции — только на уровне пакета

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

func greet(user string) {
  fmt.Println("Hello " + user)
}

func Hell2() {
  func Hello3() { // Unresolved reference 'Hello3'
     fmt.Println("from Hello2")
  }
  fmt.Println("Hello World!")
}
```

Variadic functions

```
package main

import "fmt"

func sum(nums ...int) (s int) {
  for  , num := range nums {
     s += num
  }
  return s
}

func main() {
  fmt.Println(sum(1, 2, 3)) // 5
  fmt.Println(sum())        // 0
  fmt.Println(sum(3))       // 3
}
```

Pack operator ...

Собирает параметры в слайс, а также распаковывает их

```
fmt.Println("one", "two", "three", "four")
```

```
// Println formats using the default formats for its operands and
writes to standard output.
// Spaces are always added between operands and a newline is appended.
// It returns the number of bytes written and any write error
encountered.
func Println(a ...interface{}) (n int, err error) {
  return Fprintln(os.Stdout, a...)
}
```

Только последний параметр может быть вариадическим

```
func variadic(a int, b string, s ...float64) {
  // ...
}
```

Задание 3

Реализуйте вариадическую функцию min:

https://play.golang.org/p/VdW-QBBrkhk

```
func min(nums ...int) int {
  return 0
}
```

Анонимные функции

Анонимная функция — определение функции, не связанное с идентификатором

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

```
var foo func() = func() {
  fmt.Println("Hello!")
}
foo() // Hello!
```

```
foo := func() {
  fmt.Println("Hello!")
}
foo()
```

Зачем нужны анонимные функции?

Например, компаратор для сортировки в стандартной библиотеке:

```
people := []string{"Alice", "Bob", "Dave"}
sort.Slice(people, func(i, j int) bool {
  return len(people[i]) < len(people[j])
})
fmt.Println(people) // [Bob Dave Alice]
```

Замыкания

Замыкание — это особый вид функции. Она определена в теле другой функции и создаётся каждый раз во время её выполнения. Синтаксически это выглядит как функция, находящаяся целиком в теле другой функции. При этом вложенная внутренняя функция содержит ссылки на локальные переменные внешней функции.

```go
package main

import "fmt"

func intSeq() func() int {
  i := 0
  return func() int {
     i++
     return i
  }
}
func main() {
  nextInt := intSeq()
  fmt.Println(nextInt()) // 1
  fmt.Println(nextInt()) // 2
  fmt.Println(nextInt()) // 3
  newInts := intSeq()
  fmt.Println(newInts()) // 1
}
```

Пример замыкания

Функции: сигнатуры и типы

Тип функции определяет набор всех функций с одним и тем же набором параметров и результатов (и их типов).

Неинициализированная переменная типа функции - nil.

Сигнатура - это "тип функции", примеры:

```
func()
func(x int) int
func(a, b int, z float32) bool
func(prefix string, values ...int)
```

Функции: сигнатуры и типы

```
package main

type SumFunc func(base int, arguments ...int) int

func main() {
  var summer SumFunc
  summer = func(a int, args ...int) int {
     for _, v := range args {
        a = a + v
     }
     return a
  }
  println(summer(1, 2, 3, 4)) // 10
}
```

Пример из стандартной библиотеки

```
// The HandlerFunc type is an adapter to allow the use of
// ordinary functions as HTTP handlers. If f is a function
// with the appropriate signature, HandlerFunc(f) is a
// Handler that calls f.
type HandlerFunc func(ResponseWriter, *Request)
```

Ошибки

- Ошибка — тип, реализующий интерфейс error

```
// The error built-in interface type is the conventional interface for
// representing an error condition, with the nil value representing no
error.
type error interface {
  Error() string
}
```

- Функции возвращают ошибки как обычные значения
- По конвенции, ошибка - последнее возвращаемое функцией значение
- Ошибки обрабатываются проверкой значения (и/или передаются выше через обычный return)

```go
package main

import (
  "errors"
  "fmt"
)

func greet(name string) (string, error) {
  if name == "" {
     return "", errors.New("empty name provided")
  }
  return "Hello, " + name + "!", nil
}

func main() {
  greeting, err := greet("")
  if err != nil {
     fmt.Printf("error occurred: %s\n", err)
  } else {
     fmt.Println(greeting)
  }
}
```

Примитивные ошибки

Ошибки из стандартной библиотеки

```
package errors

// New returns an error that formats as the given text.
// Each call to New returns a distinct error value even if the text is
identical.
func New(text string) error {
  return &errorString{text}
}

// errorString is a trivial implementation of error.
type errorString struct {
  s string
}

func (e *errorString) Error() string {
  return e.s
}
```

Задание 4

Модифицируйте функцию min так, чтобы она возвращала ошибку, если она была вызвана без единого аргумента

https://play.golang.org/p/EUN9J2ONOdW

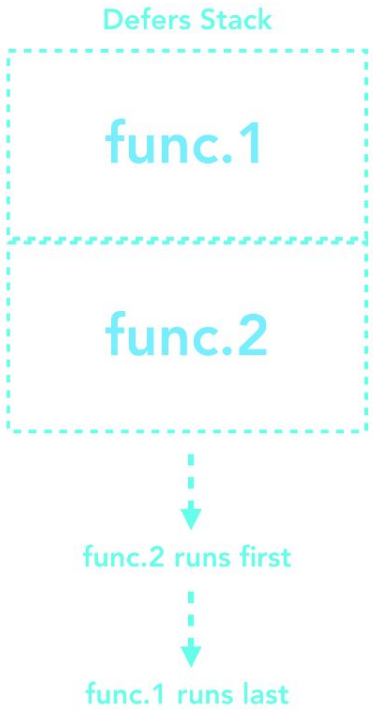
```
// TODO: implement emptyNums
var emptyNums error = nil

// TODO: implement min
func min(nums ...int) (int, error) {
  return 0, nil
}
```

Отложенное выполнение defer

```go
package main

import "fmt"

func main() {
  defer func() {
     fmt.Println("world") // потом это
  }()
  fmt.Println("hello") // сначала это
}

// hello
// world
```

Отложенные функции складываются в стек LIFO

```
func stacked() {

  defer func() {
    fmt.Println("last")
  }()

  defer func() {
    fmt.Println("first")
  }()

  // other code

  // deferred funcs run
}
```



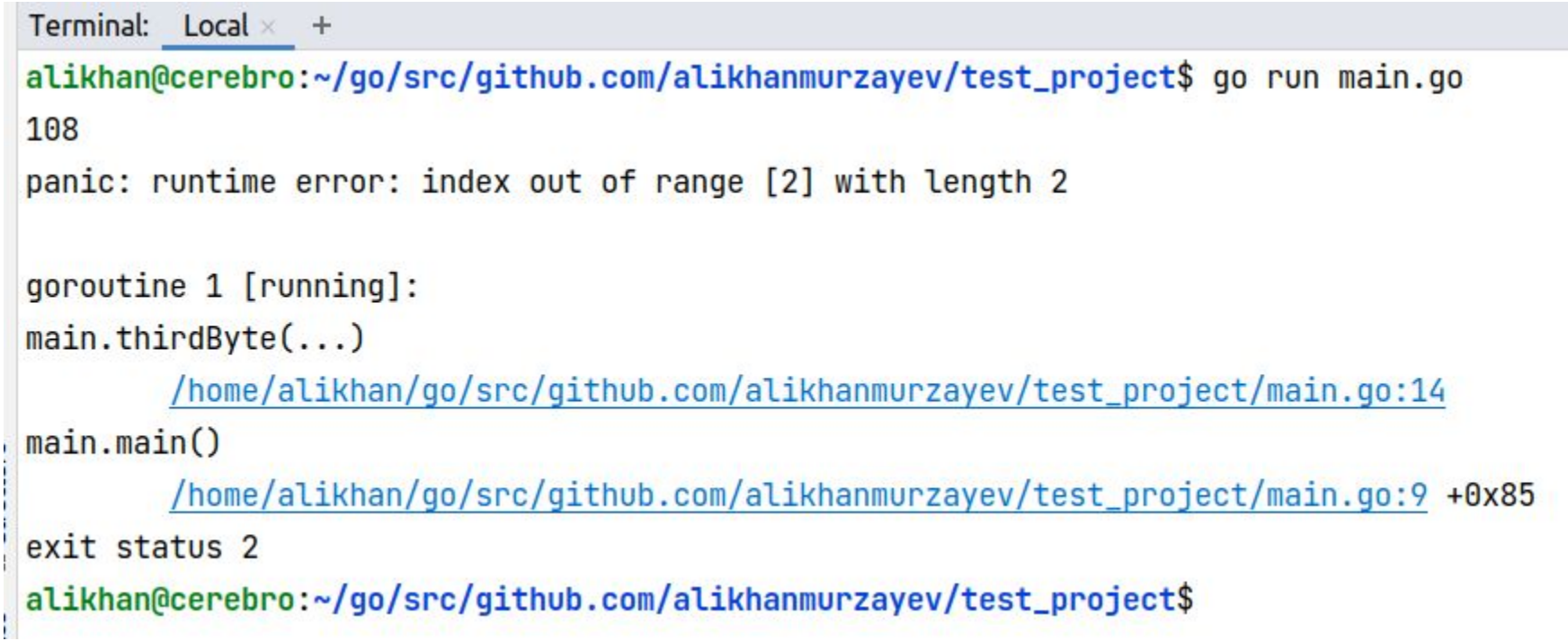
```go
package main

import (
  "log"
  "os"
)

func main() {
  f, err := os.Create("data.txt")
  if err != nil {
     log.Fatalf("could not create file: %s", err)
  }
  defer func() {
     if closeErr := f.Close(); closeErr != nil {
        log.Fatalf("could not close file: %s", err)
     }
  }()
  n, err := f.WriteString("Hello, world!")
  if err != nil {
     log.Fatalf("could not write string: %s", err)
  }
  log.Printf("wrote %d bytes", n)

}
```

Отложенное закрытие файла

Безобидный кусок дерьма кода

```
package main

import (
  "fmt"
)

func main() {
  fmt.Println(thirdByte("hello"))
  fmt.Println(thirdByte("he"))
  fmt.Println("finished successfully")
}

func thirdByte(s string) byte {
  return s[2]
}
```

Который роняет через джамбас приложение



Что такое паника и как с ней справляться?!

Что такое паника?

panic — это встроенная функция, которая останавливает обычный поток управления и начинает паниковать. Когда функция F вызывает panic, выполнение F останавливается, все отложенные вызовы в F выполняются нормально, затем F возвращает управление вызывающей функции. Для вызывающей функции вызов F ведёт себя как вызов panic. Процесс продолжается вверх по стеку, пока все функции в текущей го-процедуре не завершат выполнение, после чего аварийно останавливается программа. Паника может быть вызвана прямым вызовом panic, а также вследствие ошибок времени выполнения, таких как доступ вне границ массива.

Явный вызов паники

```
package main

import "fmt"

func main() {
  defer func() {
     fmt.Println("second")
  }()
  fmt.Println("first")
  panic("just because")
  fmt.Println("will never printed")
}
```


BLACK
FLEX
panic
recover

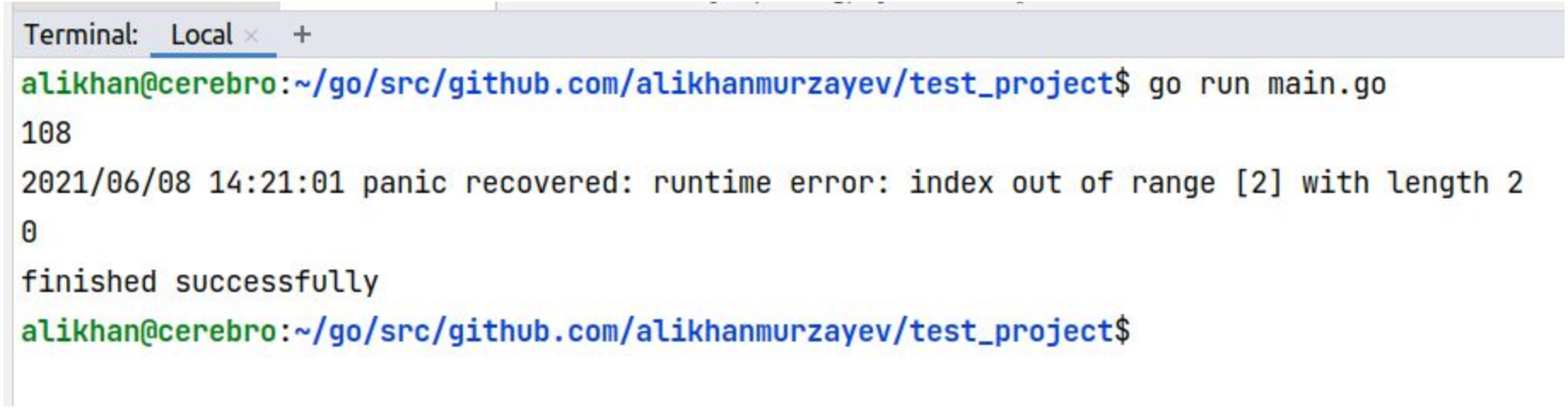
Что такое recover?

Recover — это встроенная функция, которая восстанавливает контроль над паникующей го-процедурой. Recover полезна только внутри отложенного вызова функции. Во время нормального выполнения, recover возвращает nil и не имеет других эффектов. Если же текущая го-процедура паникует, то вызов recover возвращает значение, которое было передано panic и восстанавливает нормальное выполнение.

Ловим панику

```go
package main

import (
  "fmt"
  "log"
)

func main() {
  fmt.Println(thirdByte("hello"))
  fmt.Println(thirdByte("he"))
  fmt.Println("finished successfully")
}

func thirdByte(s string) byte {
  defer func() {
     if r := recover(); r != nil {
        log.Printf("panic recovered: %v", r)
     }
  }()
  return s[2]
}
```

Программа успешно завершилась



Пример из стандартной библиотеки

```
// jsonError is an error wrapper type for internal use only.
// Panics with errors are wrapped in jsonError so that the top-level
recover
// can distinguish intentional panics from this package.
type jsonError struct{ error }

func (e *encodeState) marshal(v interface{}, opts encOpts) (err error) {
  defer func() {
     if r := recover(); r != nil {
        if je, ok := r.(jsonError); ok {
           err = je.error
        } else {
           panic(r)
        }
     }
  }()
  e.reflectValue(reflect.ValueOf(v), opts)
  return nil
}
```

Паника в коде — горе в продакшене

На этом пока всё

часть презентации была бессовестно украдена

Вопросы?

ДЗ

Все функции реализовать в одном модуле, но разных пакетах. Один исполняемый файл должен продемонстрировать работу всех функций. Скоро познакомимся с тестами, а пока что так...

1. Прочитать про блоки: https://go101.org/article/blocks-and-scopes.html.
2. Написать функцию `itoa(n int) string`.
3. Написать функцию `atoi(s string) (int, error)`.
4. Переделать функцию `reverse(s string) string` так, чтобы она работала корректно с символами, которые занимают больше одного байта (кириллица, например).
5. Написать функцию, в качестве параметра принимает путь к go-файлу. Функция должна отсортировать импорты и перезаписать файл.

ДЗ — продолжение

6. Замыкания: написать функцию `fibonacci() func() int`:

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

7. Написать функцию `runeByIndex(s *string, i *int) (rune, error)`. Код писать, придерживаясь принципа “It's easier to ask forgiveness than it is to get permission”. В случае возникновения паники необходимо завернуть ее в ошибку и вернуть из функции.