ОСНОВЫ WEB-РАЗРАБОТКИ

Лекция 5. Golang

Курс читают:

Шульман В.Д. @ShtuzerVD
Пелевина Т.В. @anivelat
Шабанов В.В. @ZeroHug

ПЛАН ЛЕКЦИИ

- Go

HELLO WORLD


```go
package main

import "fmt"

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

HELLO WORLD


```
go run main.go
```

ПЕРЕМЕННЫЕ


```
var i int
var s string
```

ПЕРЕМЕННЫЕ


```
i = 20
s = "Some String"
```

ПЕРЕМЕННЫЕ


```
var k int = 35
```

ПЕРЕМЕННЫЕ


```
var score = 10
```

ПЕРЕМЕННЫЕ


```
j := 50
str := "Some String!"
```

ПЕРЕМЕННЫЕ


```
firstName, lastName := "FirstName", "LastName"
```

ПЕРЕМЕННЫЕ


```
var (
    name = "Donald Duck"
    age  = 50
)
```

КОНСТАНТЫ


```
const PI float64 = 3.14159265359
const VALUE = 1000
```

КОНСТАНТЫ


```
const (
    PRODUCT  = "Ice Cream"
    QUANTITY = 50
)
```

ТИПЫ ДАННЫХ


```
uint8       unsigned  8-bit integers (0 to 255)
uint16      unsigned 16-bit integers (0 to 65535)
uint32      unsigned 32-bit integers (0 to 4294967295)
uint64      unsigned 64-bit integers (0 to 18446744073709551615)
int8        signed  8-bit integers (-128 to 127)
int16       signed 16-bit integers (-32768 to 32767)
int32       signed 32-bit integers (-2147483648 to 2147483647)
int64       signed 64-bit integers (-9223372036854775808 to 9223372036854775807)

float32     IEEE-754 32-bit floating-point numbers
float64     IEEE-754 64-bit floating-point numbers
```

ТИПЫ ДАННЫХ


```
complex64   complex numbers with float32 real and imaginary parts
complex128  complex numbers with float64 real and imaginary parts

byte        alias for uint8
rune        alias for int32

uint     unsigned, either 32 or 64 bits
int      signed, either 32 or 64 bits
uintptr  unsigned integer large enough to store the uninterpreted bits of a pointer value
```

ТИПЫ ДАННЫХ


```go
// Integer
var i int = 5
fmt.Println(i)

// String
var s string = "Hello World!"
fmt.Println(s)

// Float
var f float64 = 3.14159265359
fmt.Println(f)

// Boolean
var b bool = true
fmt.Println(b)
```

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ


```go
package main

import "fmt"

func main() {
    var i int = 10
    var k int = 20

    // Arithmetic Operators
    fmt.Printf("i + k = %d\n", i+k)
    fmt.Printf("i - k = %d\n", i-k)
    fmt.Printf("i * k = %d\n", i*k)
    fmt.Printf("i / k = %d\n", i/k)
    fmt.Printf("i mod k = %d\n", i%k)
}
```

```
# Output
i + k = 30
i - k = -10
i * k = 200
i / k = 0
i mod k = 10
```

ОПЕРАЦИИ СРАВНЕНИЯ


```go
package main

import "fmt"

func main() {
    var i int = 10
    var k int = 20

    // Comparison Operators
    fmt.Println(i == k)
    fmt.Println(i != k)
    fmt.Println(i < k)
    fmt.Println(i <= k)
    fmt.Println(i > k)
    fmt.Println(i >= k)
}
```

```
# Output
false
true
true
true
false
false
```

ОПЕРАЦИИ СРАВНЕНИЯ


```go
package main

import "fmt"

func main() {
    var i int = 10
    var k int = 20
    var z int = 30

    // Logical Operators
    fmt.Println(i < z && i > k)
    fmt.Println(i < z || i > k)
    fmt.Println(!(i == z && i > k))
}
```

```
# Output
false
true
true
```

ПРИМЕР МАТЕМАТИЧЕСКИХ ОПЕРАЦИЙ


```go
func main() {
    // Assignment Operators
    var x, y = 15, 25
    x = y
    fmt.Println("= ", x)

    x = 15
    x += y
    fmt.Println("+=", x)

    x = 50
    x -= y
    fmt.Println("-=", x)

    x = 2
    x *= y
    fmt.Println("*=", x)

    x = 100
    x /= y
    fmt.Println("/=", x)

    x = 40
    x %= y
    fmt.Println("%=", x)
}
```

```
# Output
=  25
+= 40
-= 25
*= 50
/= 4
%= 15
```

ФУНКЦИИ


```go
package main

import "fmt"

func main() {
    helloWorld()
}

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

ФУНКЦИИ


```go
package main

import "fmt"

func main() {
    hello("Go")
    add(20, 30)
}

func hello(x string) {
    fmt.Printf("Hello %s\n", x)
}

func add(x int, y int) {
    fmt.Println(x + y)
}
```

```
Hello Go
50
```

ФУНКЦИИ


```go
package main

import "fmt"

// Returning a single value of type int
func add(x int, y int) int {
    return x + y
}

func main() {
    // Accepting return value in variable
    sum := add(20, 30)
    fmt.Println("Sum: ", sum)
}
```

ФУНКЦИИ


```go
package main

import "fmt"

// Named return value
func getArea(l int, b int) (area int) {
    area = l * b
    return // Return without specify variable name
}


func main() {
    // Accepting a named return value
    area := getArea(10, 10)
    fmt.Println("Area: ", area)
}
```

ФУНКЦИИ


```go
package main

import "fmt"

// Returning multiple name values
func rectangle(l int, b int) (area int, parameter int) {
    parameter = 2 * (l + b)
    area = l * b
    return
}

func main() {
    // Accepting multiple return values
    area, parameter := rectangle(10, 10)
    fmt.Println("Area: ", area)
    fmt.Println("Parameter", parameter)
}
```

ФУНКЦИИ. УКАЗАТЕЛЬ


```go
package main

import "fmt"

// Passing addresses to a function
func addValue(x *int, y *string) {
    *x = *x + 5
    *y = *y + " World!"
    return
}

func main() {
    var number = 20
    var text = "Hello"
    fmt.Println("Before:", text, number)

    addValue(&number, &text)

    fmt.Println("After:", text, number)
}
```

ФУНКЦИИ


```go
package main

import "fmt"

func main() {
  func(name string) {
        fmt.Println("Hello ", name)
  }("Everyone!")
}
```

ФУНКЦИИ


```go
package main

import "fmt"

func main() {
  func(name string) {
        fmt.Println("Hello ", name)
  }("Everyone!")
}
```

ФУНКЦИИ


```go
package main

import "fmt"

// Defining a anonymous function
var (
    area = func(l int, b int) int {
        return l * b
    }
)

func main() {
    area := area(10, 10)
    fmt.Println(area)
}
```

ФУНКЦИИ


```go
package main

import "fmt"

func main() {
    l := 10
    b := 10

    // Closure functions are a special case of a anonymous function
    func() {
        var area int
        area = l * b
        fmt.Println(area)
    }()
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
package main

import (
    "fmt"
)

func main() {
    // If Statement
    x := true
    if x == true {
        fmt.Println("True")
    }
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
package main

import (
    "fmt"
)

func main() {
    // If-Else Statement
    y := 100
    if y > 80 {
        fmt.Println("Greater than 80")
    } else {
        fmt.Println("Lesser than 80")
    }
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
package main

import (
    "fmt"
)

func main() {
    // If-Elseif Statement
    grade := 5
    if grade == 1 {
        fmt.Println("You have an A")
    } else if grade > 1 && grade < 5 {
        fmt.Println("You have no A but you are positiv")
    } else {
        fmt.Println("Your grade is negativ")
    }
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
package main

import (
    "fmt"
)

func main() {
    // Switch Statement
    num := 1
    switch num {
    case 1:
        fmt.Println("One")
    case 2:
        fmt.Println("Two")
    default:
        fmt.Println("Many")
    }
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
package main

import (
    "fmt"
)

func main() {
    // Switch Statement
    num := 1
    switch num {
    case 1:
        fmt.Println("One")
    case 2:
        fmt.Println("Two")
    default:
        fmt.Println("Many")
    }
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
package main

import (
    "fmt"
)

func main() {
    // Switch Statement with multiple cases
    switch num {
    case 1, 2, 3, 4, 5:
        fmt.Println("Some")
    case 6, 7, 8, 9:
        fmt.Println("More")
    default:
        fmt.Println("Many")
    }
}
```

КОНСТРУКЦИИ ВЕТВЛЕНИЯ


```go
func main() {
    // Switch fallthrough case statement (forces the execution of the next statement)
    dayOfWeek := 3
    switch dayOfWeek {
    case 1:
        fmt.Println("Go to work")
        fallthrough
    case 2:
        fmt.Println("Buy some bread")
        fallthrough
    case 3:
        fmt.Println("Visit a friend")
        fallthrough
    case 4:
        fmt.Println("Buy some food")
        fallthrough
    case 5:
        fmt.Println("See your family")
    default:
        fmt.Println("No information available for that day.")
    }
}
```

```
# Output
Visit a friend
Buy some food
See your family
```

ЦИКЛЫ


```go
package main

import (
    "fmt"
)

func main() {
    // Basic for loop
    for i := 0; i <= 10; i++ {
        fmt.Println(i)
    }
}
```

ЦИКЛЫ


```go
package main

import (
    "fmt"
)

func main() {
    // Infinite loop
    i := 0
    for {
        fmt.Println("Hello World!")
        // Breaks/Stops the infinite loop
        if i == 10 {
            break
        }
        i++
    }
}
```

ЦИКЛЫ


```go
package main

import "fmt"

func main() {
    // Basic while loop
    x := 0
    for x < 10 {
        fmt.Println(x)
        x++
    }
}
```

ЦИКЛЫ


```go
package main

import "fmt"

func main() {
    // Do while loop
    num := 0
    for {
        // Work
        fmt.Println(num)

        if num == 10 {
            break
        }
        num++
    }
}
```

МАССИВЫ


```go
package main

import "fmt"

func main() {
    // Declaring an Array
    var intArray [5]int
}
```

МАССИВЫ


```go
package main

import "fmt"

func main() {
    // Declaring an Array
    var intArray [5]int

    // Assigning values
    intArray[0] = 10
    intArray[1] = 2

    // Accessing the elements
    fmt.Println(intArray[0])
    fmt.Println(intArray[1])
}
```

МАССИВЫ


```go
func main() {
    // Initialize Array using Array literals
    x := [5]int{0, 5, 10, 15, 20}
    var y [5]int = [5]int{0, 5, 10, 15, 20}

    fmt.Println(x)
    fmt.Println(y)
}
```

```
[0 5 10 15 20]
[0 5 10 15 20]
```

МАССИВЫ


```go
package main

import "fmt"

func main() {
    // Initializing an Array with ellipses
    k := [...]int{10, 20, 30}

    fmt.Println(len(k))
}
```

МАССИВЫ


```
for i := 0; i < len(x); i++ {
    fmt.Println(x[i])
}

// Getting index and value using range
for index, element := range x {
    fmt.Println(index, "=>", element)
}

// Only getting value using range
for _, value := range x {
    fmt.Println(value)
}

// Range and counter
j := 0
for range x {
    fmt.Println(x[j])
    j++
}
```

СРЕЗЫ (СЛАЙСЫ)


```go
func main() {
    // Create an empty slice
    var x []int
    fmt.Println(reflect.ValueOf(x).Kind())

    // Creating a slice using the make function
    var y = make([]string, 10, 20)
    fmt.Printf("y \tLen: %v \tCap: %v\n", len(y), cap(y))

    // Initialize the slice with values using a slice literal
    var z = []int{10, 20, 30, 40}
    fmt.Printf("z \tLen: %v \tCap: %v\n", len(z), cap(z))
    fmt.Println(z)

    // Creating a Slice using the new keyword
    var a = new([50]int)[0:10]
    fmt.Printf("a \tLen: %v \tCap: %v\n", len(a), cap(a))
    fmt.Println(a)
}
```

СРЕЗЫ (СЛАЙСЫ)


```go
package main

import (
    "fmt"
    "reflect"
)

func main() {
    // Add items using the append function
    var b = make([]int, 1, 10)
    fmt.Println(b)
    b = append(b, 20)
    fmt.Println(b)
}
```

СРЕЗЫ (СЛАЙСЫ)


```go
func main() {
    // Access slice items
    var c = []int{10, 20, 30, 40}
    fmt.Println(c[0])
    fmt.Println(c[0:3])
}
```

СРЕЗЫ (СЛАЙСЫ)


```go
// Copy slice into another slice
var e = []int{10, 20, 30, 40}
var f = []int{50, 60, 70, 80}
copy(e, f)
fmt.Println("E: ", e)

// Append a slice to an existing one
var g = []int{10, 20, 30, 40}
var h = []int{50, 60, 70, 80}

g = append(g, h...)
fmt.Println(g)
```

СРЕЗЫ (СЛАЙСЫ)


```go
// Copy slice into another slice
var e = []int{10, 20, 30, 40}
var f = []int{50, 60, 70, 80}
copy(e, f)
fmt.Println("E: ", e)

// Append a slice to an existing one
var g = []int{10, 20, 30, 40}
var h = []int{50, 60, 70, 80}

g = append(g, h...)
fmt.Println(g)
```

МАПА


```go
func main() {
    // Declaring empty map
    var shopingList = map[string]int{}
    fmt.Println(shopingList)

    // Initializing a map
    var people = map[string]int{"Elon": 10, "Jeff": 15}
    fmt.Println(people)

    // Map declaration using make function
    var peopleList = make(map[string]int)
    peopleList["Elon"] = 10
    peopleList["Jeff"] = 15
    fmt.Println(peopleList)
}
```

МАПА


```go
var m = map[string]string{
    "c": "Cyan",
    "y": "Yellow",
    "m": "Magenta",
    "k": "Black",
}

func main() {
    // Accessing items
    fmt.Println(m["c"])

    // Adding items
    m["b"] = "black"
    fmt.Println(m)

    // Updating items
    m["y"] = "lemon yellow"
    fmt.Println(m)

    // Deleting items
    delete(m, "b")
    fmt.Println(m)
}
```

```
Cyan
map[b:black c:Cyan k:Black m:Magenta y:Yellow]
map[b:black c:Cyan k:Black m:Magenta y:lemon yellow]
map[c:Cyan k:Black m:Magenta y:lemon yellow]
```

МАПА


```go
var m = map[string]string{
    "c": "Cyan",
    "y": "Yellow",
    "m": "Magenta",
    "k": "Black",
}

func main() {
    // Iterating over a map
    for k, v := range m {
        fmt.Printf("Key: %s, Value: %s", k, v)
    }
}
```

```
Key: m, Value: Magenta
Key: k, Value: Black
Key: c, Value: Cyan
Key: y, Value: Yellow
```

МАПА


```
func main() {
    // Test if an item exists
    c, ok := m["y"]
    fmt.Println("\nc: ", c)
    fmt.Println("ok: ", ok)
}
```

СТРУКТУРЫ


```go
package main

import "fmt"

// Declaring a Struct
type Animal struct {
    name   string
    weight int
}

func main() {
}
```

СТРУКТУРЫ


```go
package main

import "fmt"

// Declaring a Struct
type Animal struct {
    name   string
    weight int
}

func main() {
    // Creating an instance of a struct
    var dog Animal
    dog.name = "Dog"
    dog.weight = 40
    fmt.Println(dog)
```

СТРУКТУРЫ


```go
// Creating an instance using struct literate
var cat = Animal{name: "Cat", weight: 5}
fmt.Println(cat)

// Creating an instance using the new keyword
var bird = new(Animal)
bird.name = "Bird"
bird.weight = 1
fmt.Println(bird)

// Creating an instance using the pointer address operator
var monkey = &Animal{name: "Monkey", weight: 10}
fmt.Println(monkey)
```

```
{Cat 5}
&{Bird 1}
&{Monkey 10}
```

СТРУКТУРЫ. МЕТОДЫ


```go
type Config struct {
    Env   string
    Proxy ProxyInfo
}

type ProxyInfo struct {
    Address string
    Port    string
}

func (conf Config) ConfInfo() string {
    fmt.Println("Env: ", conf.Env)
    fmt.Println("Proxy: ", conf.Proxy)
    return "----------------------"
}
```

СТРУКТУРЫ. МЕТОДЫ


```go
func main() {
    c := &Config{
        Env: "DEBUG:TRUE",
        Proxy: ProxyInfo{
            Address: "addr",
            Port:    "port",
        },
    }

    fmt.Println(c.ConfInfo())
}
```

ИНТЕРФЕЙСЫ


```go
// Defining a interface
type User interface {
    PrintName(name string)
}

type Vehicle interface {
    Alert() string
}

func main() {

}
```

ИНТЕРФЕЙСЫ


```go
// Defining a interface
type User interface {
    PrintName(name string)
}

type Vehicle interface {
    Alert() string
}

func main() {

}
```

ИНТЕРФЕЙСЫ


```go
// Defining a interface
type User interface {
    PrintName(name string)
}

type Vehicle interface {
    Alert() string
}

func main() {

}
```

ИНТЕРФЕЙСЫ


```go
// Create type for interface
type Usr int

type Car struct{}

// Implement interface function in type
func (usr Usr) PrintName(name string) {
    fmt.Println("User Id:\t", usr)
    fmt.Println("User Name:\t", name)
}

func (c Car) Alert() string {
    return "Hup! Hup!"
}
```

ИНТЕРФЕЙСЫ


```go
func main() {
    var user1 User
    user1 = Usr(1)
    user1.PrintName("Gabriel")

    c := Car{}
    fmt.Println(c.Alert())

    Print(20)
}
```

ИСТОЧНИКИ

- **An Introduction to Golang**
  https://gabrieltanner.org/blog/an-introduction-to-golang/

СПАСИБО ЗА ВНИМАНИЕ :3

01.10.2024