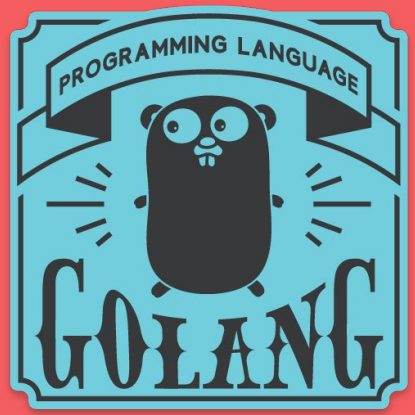
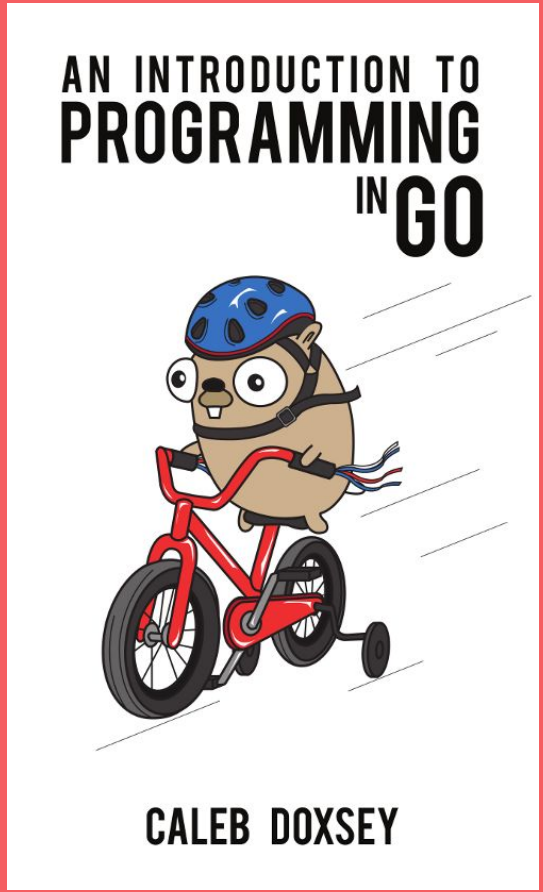


Introdução

Thanks



https://www.golang-book.com/books/intro

Agenda

- Ambiente
- Hello World
- Tipos Primitivos
- Variáveis & Constantes
- Estrutura de Controle
- Estrutura de Dados
- Funções
- Type
- Ponteiro
- Package
- Concorrência
- Testes

Ambiente

- Configuração
- Editores

Ambiente SDK

Download GO SDK

- https://golang.org/dl/

Definir as Variáveis de Ambiente Globais

- *export GOROOT=/usr/local/go*
- *export PATH=$PATH:$GOROOT/bin*

Mais informações: https://golang.org/doc/install

Ambiente de Desenvolvimento

Criar o Diretório Go de Desenvolvimento

- mkdir ~/home

Criar os Diretórios Principais

- *mkdir ~/go/lib*
- *mkdir ~/go/bin*
- *mkdir ~/go/src*

Definir as Variáveis de Ambiente

- *export GOPATH=~/go*
- *export PATH=$PATH:$GOPATH/bin*

Estrutura de Diretórios

lib

- bibliotecas: dll, .so

bin

- executáveis

src

- código fonte

Editores

- Visual Studio Code
- Vim
- Atom

Hello Word

Hello World

Criar um novo projeto com o nome "test"

- mkdir ~/go/src/test

Criar um arquivo "oops.go" dentro do projeto *test* com o conteúdo:

```
package main

import "fmt"

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

Tipos Primitivos

- Números
- Strings
- Boolean

Números

uint, uint8, uint16, uint32, uint64

- {x | x ∈ ℕ} *naturais*
- *0, 1, 2, 3, …*

int, int8, int16, int32, int64

- {x | x ∈ ℤ} *inteiros*
- …, -2, -1, 0, 1, 2, …

float, float32, float64

- {x | x ∈ ℝ} *reais*
- …, -2.x, -1.x, 0, 1.x, 2.x, …

complex64, complex128

- {x | x ∈ ℂ} *complexo*
- "math/cmplx"

Números - Operações

Adição

- fmt.Println("1 + 1", 1 + 1)

Subtração

- fmt.Println("1 - 1", 1 - 1)

Multiplicação

- fmt.Println("1 * 1", 1 * 1)

Divisão

- fmt.Println("1 / 1", 1 / 1)
- **Divisão por 0**
  - **panic** int
  - **NaN** float*

String

- “asfsaf”
- fmt.Println("abc")
- fmt.Println("abc" + "def")
- fmt.Println("abc"[0])
    - fmt.Printf("%c", "asd"[0])

- fmt.Println("abc" + 1)
    - fmt.Printf("%s %d", "asd", 1)

Boolean

true & false

- && *and*
- || or
- ! not

Variável & Constantes

- Declaração
- Nomenclatura

Variáveis

- Tipada
- Não é obrigatório informar o tipo durante a declaração
- Multi declaração
- *Case Sensitive* [Name != name]
- Toda variável declarada ***tem que ser utilizada***

Variáveis

EX:

- ***var*** name string = "teste"
- ***var*** name = "teste"
- name := "teste"
- name, age := "name", 18
- ***var*** (
  name = "name"
  age = 18
  email = "test@test.com"
)

Constantes

- Tipada
- Não é obrigatório informar o tipo durante a declaração
- Multi declaração
- *Case Sensitive* [Name != name]
- ~~Toda constante declarada ***tem que ser utilizada***~~
- Devem começar com a palavra ***const***
- Seu valor não pode ser alterado

Constantes

EX:

- ***const*** name string = "teste"
- ***const*** name = "teste"
- ***const*** (
  name = "name"
  age = 18
  email = "test@test.com"
)

Nomenclatura

[a-zA-Z_][\w]*

- Deve começar com letras ou ‘_’
- Pode conter números, letras e ‘_’

Estrutura de Controle

- for
- if
- switch

for

- Estrutura de repetição

Tipos

- for ***condição*** {
  ...
  }
- for ***contador*** ; ***condição***; ***increment*** {
  ...
  }
- for ***contador*** , ***elemento*** := range ***array*** {
  ...
  }

if

- Estrutura Condicional

EX

- If x <= 3 {
  ...
  }
- If x == 1 {
  fmt.Println("um")
  } else if x == 2 {
  fmt.Println("dois")
  } else {
  fmt.Println("oops")
  }

switch

- Estrutura Condicional

EX:

```
switch x {
case 1: fmt.Println("um")
case 2: fmt.Println("dois")
default: fmt.Println("oops")
}
```

Estrutura de Dados

- array
- slice
- map

array

- Sequência numerada de um mesmo tipo
- Inicia no index 0

Inicialização

- var matriz [5]int
  matriz[0] = 1
  matrix[1] = 2
  ...
  matriz[4] = 5
  ~~matriz[5] = 6~~ // não compila

- matriz := [5]int {1, 2, 3, 4, 5}

array

acesso

- matriz := [5]int {1, 2, 3, 4, 5}
  fmt.Println(matriz[3])

  ~~fmt.Println(matriz[5])~~ // não compila

Slice

- Podemos considerar um slice “como um array onde o tamanho não é predefinido”

Inicialização

- matriz := **make**([]int, 0)
- matriz := **make**([]int, 3, 3)
- matriz := []int{1, 2, 3}
- array := [4]int {0,1,2,3}
  matriz = array[1:3]

Slice - Métodos

append(slice_original, ... elementos)

- Retorna um novo slice com os elementos do slice informado e os elementos.
- Os elementos são adicionados no final do slice
  - matriz = append(matriz, 1, 2)
  - ~~append(matriz, 1, 2)~~

copy(slice_destino, slice_origem)

- Copia os dados do slice origem para o slice destino
  - copy(slice1, slice2)

slice

acesso

- array := [4]int {0,1,2,3}
  matriz = array[:]

  ~~fmt.Println(matriz[5])~~ // panic

Map

- Uma coleção baseada em chave valor.
- Não ordenado

Inicialização

var dict **map [*tipo_chave*] *tipo_valor***

- var dict map[string]int
  dict = make(map[string]int)

- dict := make(map[string]int)

Map

acesso

```
dict := make(map[string]int)
dict["test"] = 33                    // armazena o valor 33 utilizando a chave “test”

value, contains  =  dict["test"]     // acessa o valor (33) pela chave “test”

dict["test"] = 10                    // substitui o valor 33 por 10

for key := range mymap {             // intera nas chaves do map
}
```

Map

funções

delete(map, chave)

EX:

```
dict := make(map[string]int)
dict["test"] = 33

delete(dict, "test")                    // remove o valor 33
```

Global

funções

len(coleção)

EX:

```
slice := make([]int, 0)
len(slice)              // retorna a quantidade de elementos do slice

dict := make(map[string]int)
len(dict)               // retorna a quantidade de elementos do map
```

Funções

- retorno
- varargs
- closure
- defer, panic, recover

Funções

func *func_nome(* **arg type, ...)** **type**

EX:

```
func sum(a int, b int) int {
        return a + b
}

func sum(a int, b int) (result int) {
        result = a + b
        return
}
```

Retorno

func *func_nome()* **(r1, r2, ..., rn)**

EX

```
func div(a float32, b float32) float32 {
        return a/b
}
```

Retorno

func *func_nome()* **(r1, r2, ..., rn)**

EX

```
func div(a float32, b float32) (float32, error) {
        if (b == 0) {
                return 0, errors.New("divisao por zero")
        }
        return a/b, nil
}
```

Varargs

func *func_nome(**name ...type**)* type

- Possibilita passar ‘n’ valores do mesmo tipo, onde o valor de ‘n’ é definido por quem chama o método
- ***TEM que ser o último argumento da função***

EX

```
func sum(values ... int) (count int) {
	for _, value := range values {
		count += value
	}
	return
}
```

Varargs

func *func_nome(**name ...type**)* type

- Possibilita passar ‘n’ valores do mesmo tipo, onde o valor de ‘n’ é definido por quem chama o método
- ***TEM que ser o último argumento da função***

EX

```
func sum(mandatory  int, values ... int) int {
        count := mandatory
        for _, value := range values {
                count += value
        }
        return count
}
```

Retorno

func *func_nome()* **(r1, r2, ..., rn)**

EX

```
func div(a float32, b float32) (result float32, err error) {
        if (b == 0) {
                err = errors.New("divisao por zero")
                return
        }
        result = a/b
        return
}
```

Closure

- Closures (fechamentos) são funções que se referem a variáveis livres (independentes).

- Em outras palavras, a função definida no closure "lembra" do ambiente em que ela foi criada.

“https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Closures, 2018”

Closure

```
func makeEvenGenerator() func() uint {
        i := uint(0)

        return func() (ret uint) {
                ret = i
                i += 2
                return
        }
}
```

defer, panic, recover

defer

- Executado após o método finalizar

panic

- Lança um erro de tempo de execução
- Não deveriam ser tratados estes tipos de erros

recover

- Possibilita se recuperar de um ***panic***

defer

defer *metodo()*
EX:

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

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

...
defer last()
first()
```

panic

panic("mensagem"**)**

EX:
panic("ooops!!!")
fmt.Println("nunca serão")

recover

defer func() {
    recover() string
}()

EX:

```
defer func() {
    msg := recover()
    fmt.Println(msg)
}()

panic("ooops!!!")
fmt.Println("nunca serão")
```

Type

- Struct & methods
- Herança & Polimorfismo
- Interface
- *

Struct

- ***Struct*** é um novo tipo definido pelo usuário
- Podem conter outros tipos em sua estrutura, a qual chamamos de “atributos”

```
type nome struct {
}

type nome struct {
        field1 tipo1
        field1 tipo2
        ...
        field_n tipo_n
}
```

Struct

EX:

```
type User struct {
        Name            string
        Email           string
        Password        string
}

var user User
user := User{}
user := User{"name", "email", "password"}
user := User{Email: "email", Name: "name"}
```

Methods

- Possibilita adicionar comportamento nas estruturas
- Os métodos podem utilizar dos atributos da estrutura da qual ele faz parte

func (variable type) methodName(args ... type) return {}

EX:

```
func (user User) String() {
	return fmt.Sprintf("Name: %s, Email:%s", user.Name, user.Email)
}
```

“Herança” & Polimorfismo

Herança

- Capacidade de uma classe estender os comportamentos da classe estendida

Polimorfismo

- Capacidade da classe mais concreta poder definir o comportamento dos métodos das classes mais abstratas.

“Herança”

```
type TypeChild struct {
        TypeFather
}
```

EX:

```
type Admin struct {
        User
}

admin := Admin{}
admin := Admin{ User{} }
admin := Admin{ User: User{} }
```

“Herança”

var user User
user = Admin {}

“Herança”

```
var user User
user = Admin{}          // erro de compilação
```

“Herança”

```
var user User
user = Admin{}          // erro de compilação
user = Admin{}.User
```

“Herança”

```
var user User
user = Admin {}          // erro de compilação
user = Admin{}.User
```

OBS: Em go a struct herda as ações mas não o tipo. Um Admin não é visto como um tipo User.

Polimorfismo

```
type User struct {}
func (user User) Who() string {
        return "user"
}

type Admin struct {
        User
}
func (admin Admin) Who() string {
        return "admin"
}

admin.Who()         // retorna "admin"
```

Multi “Herança”

```
type TypeChild struct {
        TypeFather1
        TypeFather2
        ...
        TypeFatherN
}
```

EX:

```
type Admin struct {
        User
        Animal
}
```

Multi “Herança” & Polimorfismo

```
type User struct {}
func (u User) Who() string {
        return "user"
}

type Animal struct {}
func (u Animal) Who() string {
        return "animal"
}

type Admin struct {
        User
        Animal
}

Admin{}.Who()
```

Multi “Herança” & Polimorfismo

```
type User struct {}
func (u User) Who() string {
        return "user"
}

type Animal struct {}
func (u Animal) Who() string {
        return "animal"
}

type Admin struct {
        User
        Animal
}

~~Admin{}.Who()~~              // erro de compilacao
```

Multi “Herança” & Polimorfismo

```
type User struct {}
func (u User) Who() string {
        return "user"
}

type Animal struct {}
func (u Animal) Who() string {
        return "animal"
}

type Admin struct {
        User
        Animal
}

~~Admin{}.Who()~~              // erro de compilacao
Admin{}.User.Who()
Admin{}.Animal.Who()
```

Interfaces

- É o maior tipo de abstração
- São tipos que não podem ser instanciados
- Uma vez que um outro tipo “implementa” a interface, este tipo pode ser visto como o mesmo da interface implementada
- Um tipo pode implementar várias interfaces
- Uma interface pode “herdar” outra interface

Interfaces

type name **interface** {
    func1
    func2
    ...
    funcN
}

EX:

type Inter interface {
    MyAction()
}

Como “Implementar” uma Interface?

```
type Inter interface {
        Action()
}

type Concrete struct {}
func (c Concrete) Action() {}

var myInter Inter
myInter = Concrete{}
```

Ponteiro

- &
- *
- new

Package

- Declaração
- Utilização
- Privado/Público

Package

- Modularização
- Reuso

Declaração

package packageName

EX:

package map
package ui

Utilização

impot *packagePath/**packageName***
packageName.PackageContent

EX:

test/bla/oops.go
package bla
type Oops struct {}

test/main.go
package main
Import "test/bla"

oops := bla.Oops

Privado / Público

Privado

- Qualquer declaração: variável, função, método, atributo, serão ditos privados quando o nome do mesmo começar com uma letra ***minúscula***.
- Por privado entende-se que apenas quem estiver no mesmo *package* poderão utilizar os dados declarados.

Público

- Qualquer declaração: variável, função, método, atributo, serão ditos privados quando o nome do mesmo começar com uma letra ***maiúscula***.
- Por público entende-se que diferente parte do código, mesmo de pacotes diferentes, poderão utilizar os dados declarados.

Privado x Público

EX:

```
package test
type Test struct {          // público     pode ser utilizado em qualquer parte do código
        name                // privado     pode ser utilizado apenas dentro do pacote
        Email               // público     pode ser utilizado em qualquer parte do código
}
```

Concorrência

- Goroutines
- Chanels

Goroutine

- Executa funções de forma concorrente
- Goroutine são ***threads*** no SO

Goroutine

go function()

EX:

```
func print(number int) {
	fmt.Println(number)
}

go print(1)
go print(2)
go print(3)
```

Não tem como definirmos em qual ordem serão executadas as funções acima

Channels

- Prover uma forma de comunicação entre duas ou mais Goroutines
- Sincroniza a execução

Channels

```
func pinger(c chan string) {
	for i := 0; ; i++ {
		c <- "ping"
	}
}

func printer(c chan string) {
	for {
		msg := <- c
		fmt.Println(msg)
		time.Sleep(time.Second * 1)
	}
}

func main() {
	var c chan string = make(chan string)
	go pinger(c)
	go printer(c)
	var input string
	fmt.Scanln(&input)
}
```

Testes

Testes

- Baseado em TDD
- Permite realizar testes funcionais

Testes

file_**test**.go

```
func TestName(t *testing.T) {
}
```

Testes

div.go

```
package main

import "errors"

var nanErr = errors.New("not a number")
func div(a float32, b float32) (float32, error) {
	if b == 0 {
		return 0, nanErr
	}
	return a/b, nil
}
```

Testes

div_test.go

```
package main

import "testing"

func TestShouldReturnNanErrIfBIsZero(t *testing.T) {
	_, err := div(1,0)
	if err != nanErr {
		t.Fail()
	}
}
```

Desafio

Criar Usuário

Contexto

Uma faculdade precisa fazer o controle acadêmico dos alunos. Qual o coeficiente de rendimento de cada aluno, matérias cursadas, notas de cada matérias, dentre outros.

Fora do Escopo

- Cadastrar administradores
- Cadastrar Professores
- Cadastrar alunos

Descrição

Eu como administrador do sistema
Desejo ser capaz de cadastrar pessoas da secretaria utilizando o nome, login e senha
Para serem capazes de utilizar o sistema

Criar Usuário

Critérios de Sucesso

1
- *Dado que* um dos administradores do sistema
- *Quando* adicionar as informações nome, login e senha de uma pessoa da secretaria
- *Então* o sistema deve informar que o usuário foi criado

2
- *Dado que* um dos administradores do sistema
- *Quando* adicionar as informações nome, login e uma senha em branco
- *Então* o sistema deve informar que não foi possível criar o usuário por conta da senha

3
- *Dado que* um dos administradores do sistema
- *Quando* adicionar as informações nome, senha e um login em branco
- *Então* o sistema deve informar que não foi possível criar o usuário por conta do login

4
- *Dado que* um dos administradores do sistema
- *Quando* adicionar as informações nome, senha e um login já utilizado
- *Então* o sistema deve informar que não foi possível criar o usuário por conta do login

Login

Contexto

Uma faculdade precisa fazer o controle acadêmico dos alunos. Qual o coeficiente de rendimento de cada aluno, matérias cursadas, notas de cada matérias, dentre outros.

Fora do Escopo

- Verificar usuários bloqueados

Descrição

Eu como um usuário do sistema
Desejo poder acessar as funcionalidades do sistema
Para realizar as minhas atividades da secretaria

Login

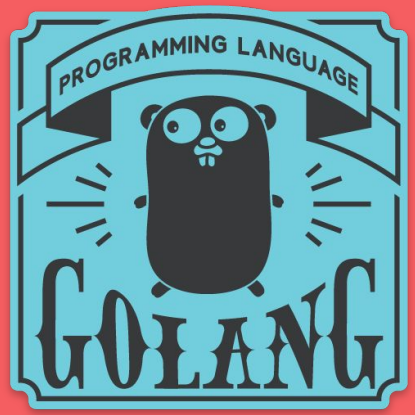
Critérios de Sucesso

1

- *Dado que* um usuário do sistema
- *Quando* informar as credenciais válidas
- *Então* o sistema deve permitir que o usuário possa utilizar as demais funcionalidades

2

- *Dado que* um usuário do sistema
- *Quando* informar sua credencial inválida
- *Então* o sistema deve informar informar que a credencial não é válida e não deve permitir acesso às demais funcionalidades



Thanks!!!