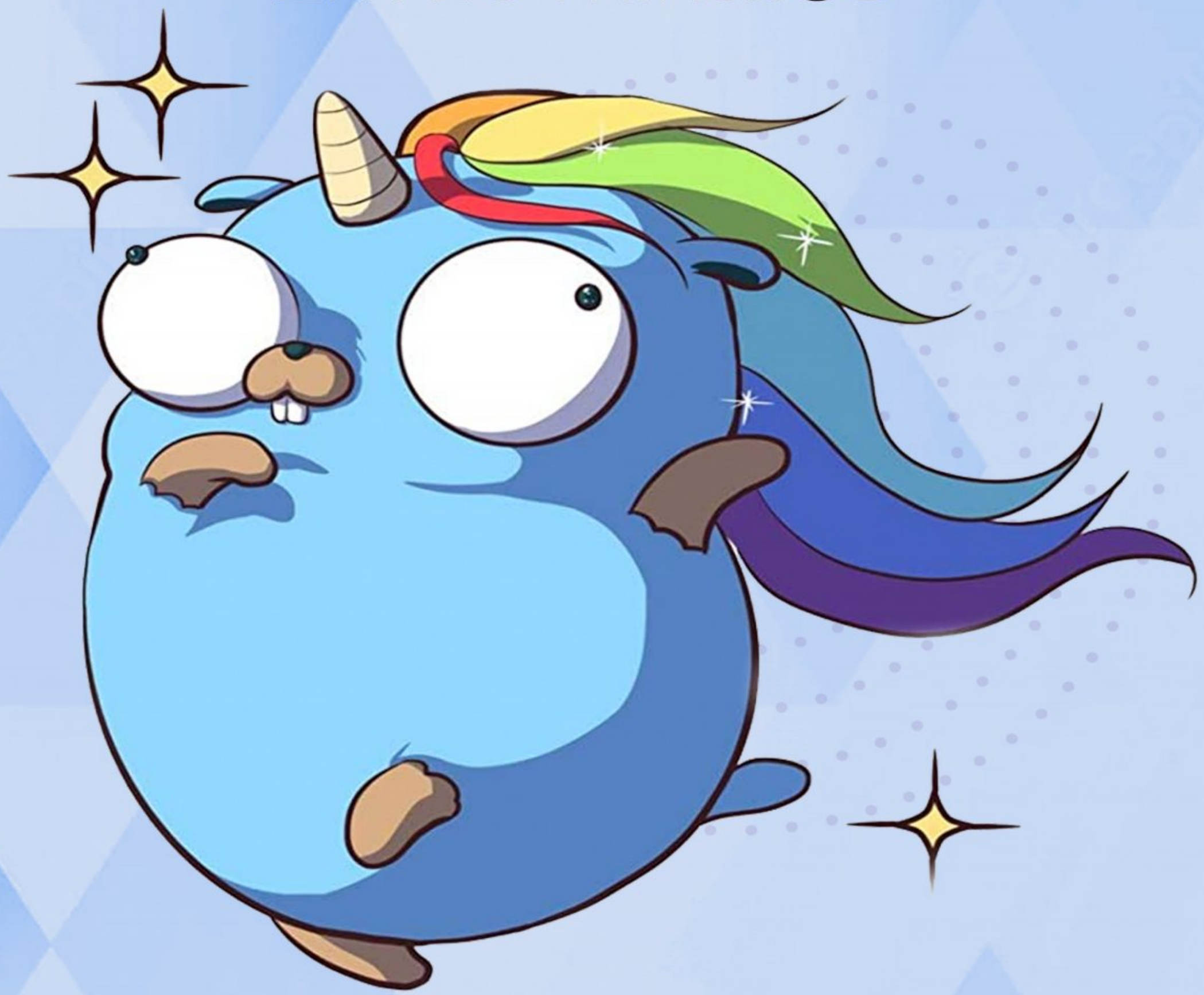
Wesley Willians

FullCycle

Guia rápido

Golang

Em 20 minutos



www.fullcycle.com.br

GoLang em 20 Minutos

Guia Rápido

Full Cycle

Esse livro está à venda em http://leanpub.com/golang-em-20minutos

Essa versão foi publicada em 2023-07-05

Leanpub

Esse é um livro Leanpub. A Leanpub dá poderes aos autores e editores a partir do processo de Publicação Lean. Publicação Lean é a ação de publicar um ebook em desenvolvimento com ferramentas leves e muitas iterações para conseguir feedbacks dos leitores, pivotar até que você tenha o livro ideal e então conseguir tração.

Conteúdo

Sobre esse guia

Esse é um guia simplificado para te ajudar a iniciar com a linguagem Go.

Ele foi criado durante a Maratona Full Cycle[1]. Um evento online e 100% gratuito que ajuda desenvolvedores a se tornarem um Full Cycle Developer, ou seja, um desenvolvedor que consegue dominar de ponta a ponta o processo de desenvolvimento de um software, desde arquitetura até deploy e o monitoramento.

Meu nome é Wesley Willians[2] e criei esse guia em uma manhã com muito carinho para você. Se você quer começar no mundo Go, sem dúvidas ele te ajudará:

- Entender os fundamentos da linguagem
- Compreender a sintaxe
- Entender os recursos básicos e suas diferenças de uma linguagem tradicional orientada a objetos.

  Pretendo aos poucos adicionar informações mais avançadas e abrangentes da linguagem nesse dia, porém, para você começar hoje, ele já é o suficiente.

Canal do Telegram

Eu super recomendo que você entre para meu Canal do Telegram[3]. Nele, você terá dicas e insights diários sobre Go Lang e também sobre o mundo do Full Cycle Development.

https://t.me/devfullcycle[4]

[1] http://pages.fullcycle.com.br/maratona-fullcycle
[2] https://www.instagram.com/devfullcycle/
[3] https://t.me/devfullcycle
[4] https://t.me/devfullcycle

Dicas para começar com a linguagem

Site da Go Lang

A linguagem Go possui um ótimo web site. Ele é extremamente documentado, possui tutoriais de iniciação, além de fornecer um editor online para você simular seus programas.

Links úteis:

- Site oficial da Linguagem[5]
- Tour interativo na linguagem[6]
- Playground[7]

Use e abuse do Playground. Ele vai te ajudar a programar com a linguagem sem ao menos você precisar instalá-la em seu computador.

Instalação

O processo de instalação da Go Lang é extremamente simples. Todavia, você precisa ficar atento, principalmente se você utiliza Windows em relação as variáveis de ambiente.

Qual IDE utilizar?

A Go Lang já está bem difundida, logo, a maioria dos editores e IDEs já a suportam.

De qualquer forma, a minha recomendação é que você utilize o VSCode ou mesmo a GoLand (IDE não gratuita da Jetbrains - é a que eu utilizo).

---

[5] https://golang.org/
[6] https://tour.golang.org/welcome/1
[7] https://play.golang.org/

Fundamentos

Função “main”

A Linguagem Go, assim como Java, precisa de pelo menos uma função “main” para o que o programa seja executado.

Segue a sintaxe de funções na Go Lang.

```
1  package main
2
3  func main() {
4
5  }
```

Pacotes

Como visto acima, também é obrigatório definirmos o nome de um pacote para que possamos utilizar a linguagem. Os pacotes facilitam muito o processo de separação de responsabilidades do programa. Eles são análogos aos pacotes do Java ou Namespaces do C#, PHP, etc.

Biblioteca padrão

A Go Lang possui uma biblioteca padrão extremamente completa, ou seja, um conjunto de pacotes que tem o objetivo de facilitar o processo de desenvolvimento. A biblioteca padrão possui recursos de rede, http, serialização, sistema de templates, SQL e muito mais.

É extremamente comum notarmos na comunidade Go a baixa adoção de “frameworks completos”, uma vez que na maioria das vezes a biblioteca padrão já supre muito diversas necessidades básicas.

Desenvolvedores Go, de forma geral são muito “puristas”, ou seja, evitam usar pacotes externos por qualquer razão.

Execução e Build

Para que você possa executar um programa desenvolvido em go, basta ter o runtime do Go instalado e executar:

```
go run arquivo.go
```

A Go Lang é uma linguagem compilada, logo, para que você gere o arquivo compilado para ser utilizado, basta executar:

```
go build arquivo.go
```

Outra grande vantagem da linguagem é que ela é multiplataforma, ou seja, podemos compilar um programa feito em Go para qualquer sistema operacional.

Exemplo:

```
GOOS=linux go build main.go
```

Sintaxe geral da linguagem

Variáveis

Declaração e atribuições de variáveis:

Você pode iniciar uma variável em qualquer contexto na Go Lang. Dentro ou fora de uma função.

```
var b int
b = 22
var c, d string = "Hello", "World"
```

Você pode apenas pode declarar e atribuir valor a uma variável de uma única vez se você estiver dentro do escopo de uma função.

```
func main() {
        a := 10
}
```

Principais tipos de variáveis

```
a := 10
b := "Hello"
c := 10.33
d := false
e := 'W'
f := `hahahaha
      Pegadinha do Malando
      = )
      `

// Imprime valores
fmt.Printf("%v \n",a)
fmt.Printf("%v \n",b)
fmt.Printf("%v \n",c)
fmt.Printf("%v \n",d)
fmt.Printf("%v \n",e)
```

```
17  fmt.Printf("%v \n",f)
18
19  // Imprime tipos
20  fmt.Printf("%T \n",a) // int
21  fmt.Printf("%T \n",b) // string
22  fmt.Printf("%T \n",c) // float64
23  fmt.Printf("%T \n",d) // bool
24  fmt.Printf("%T \n",e) //int32
25  fmt.Printf("%T \n",f) // string
```

Escopo

A Go Lang possui escopos muito bem delimitados. Todas as declarações globais, ou seja, fora de uma função, podem ser acessadas dentro do mesmo pacote. O mesmo acontece com funções. Todas as funções declaradas em um pacote, mesmo que estejam declaradas em arquivos diferentes, poderão ser acessadas de dentro de um mesmo pacote.

Variáveis declaradas dentro de uma função, fazem parte do escopo dessa mesma função.

Visibilidade

Quando uma função ou mesmo um atributo de uma struct (falaremos sobre structs em breve) estiver com com a primeira letra maiúscula, significa que ela pode ser acessada de outro pacote.

Exemplo:

```
1   /// arquivo exemplo.go
2   package exemplo
3
4   func PrintX() {
5
6   }
7
8   /// arquivo main.go
9
10  package main
11
12  import "exemplo"
13
```

```
14 func main() {
15   exemplo.PrintX()
16 }
```

A função PrintX só pode ser executada pelo fato de ela iniciar com o P maiúsculo.

Blank identifier

A Go Lang não permite declararmos uma variável e não utilizá-la. Porém, por exemplo, em alguns momentos precisamos retornar valores na chamada de uma função. Ex:

```
1 res, err = http.Get("http://fullcycle.com.br") // função retorna 2 valores
```

Se a requisição acima conseguir ser executada com sucesso, a variável "err" estará vazia e não terá sido utilizada. Logo, o programa não poderá ser compilado.

Normalmente erros são tratados da seguinte forma:

```
1 if err != nil {
2   // qualquer coisa ;)
3 }
```

Agora, e se nesse caso não quisermos fazer o tratamento desse erro?

É nessa hora que entra o famoso "Blank identifier". Ele nada mais é um underline que faz com que o Go ignore o valor vazio que tenha sido atribuido ou não a ele.

Exemplo:

```
1 res, _ = http.Get("http://fullcycle.com.br")
```

Nesse caso, o resultado de erro será ignorado, e o programa pode seguir normalmente.

Constantes

Uma constante basicamente é uma "variável" que não pode ter seu valor alterado. Na Go Lang, você precisa declarar uma constante utilizando apenas o "=" e não ":=".

A constante pode ser definida de forma global em seu pacote ou mesmo de forma local em uma função.

Exemplos de declarações:

```
const xyz int = 222
const x = 10

const (
  aa string = "x"
  bb = 66
  cc int = 567
)
```

O princícpio de visibilidade também é aplicado nas constantes.

Ponteiros e Endereçamento de memória

Na Go Lang, você tem a possibilidade de ter acesso direto ao endereçamento de memória em que um um valor é gravado.
Exemplo:

```
x := 10
fmt.Println(&x) // 0xc82000a288
```

O hexadecimal acima é o endereçamento de memória cuja variável x está sendo apontada. Para isso, tivemos que adicionar o "&" logo antes da variável.

Nesse ponto, podemos criar uma nova variável e apontá-la exatamente para o mesmo endereçamento de x.

```
y := &x
fmt.Println(y) // 0xc82000a288
```

Se em algum momento houver a necessidade de exibirmos o valor que foi atribuido para o endereçamento na memória através da variável y, basta adicionarmos "*" na frente da variável.

```
fmt.Println(*y) // 10
```

Através de pointeiros, você pode alterar o valor atribuído na memória.

```
*y = 20
fmt.Println(x) // 20
```

Perceba que o valor de x foi alterado, pois o ponteiro de Y alterou tal valor.

Um ponto que não podemos deixar de lado é que também podemos declarar uma variável definindo um tipo.

```
var z *int = &x
fmt.Println(z) // 0xc82000a288
fmt.Println(*z) // 20
```

Lembrando que podemos trabalhar com ponteiros dentro de funções.

```
func xpto(a *int) int {
 *a = 100
 return *a
}

b := 10
xpto(&b) // 100
fmt.Println(b) // 100
```

Perceba que o valor de b foi alterado pelo fato de ele ter sido passado por parâmetro na função.

Funções

Apesar de estarmos utilizando funções nos diversos exemplos, nessa seção, falaremos disso com um grau maior de profundidade.

A sintaxe de uma função é a seguinte:

```
func funcName(a int) int {

}
```

No caso acima, “funcName” é o nome da função, “a” que é do tipo inteiro é o parâmetro de entrada e “int” é o tipo de retorno.

Existe uma leve sutileza quando trabalhamos com funções em Go, veja no exemplo a seguir.

```
1  func namedReturn(a string) (x string) {
2    x = a
3    return
4  }
```

Pergunta, qual é o retorno dessa função?

Nesse caso, mesmo não estando informando especificamente qual variável devemos retornar após o "return", verifique que na definição da função, estamos especificando a variável "x", logo, nesse caso, por padrão a função retornará o valor de "x = a".

Retorno múltiplo de valores

Na Go Lang, você pode retornar mais do que um valor em uma função.

```
1  func Xpto(a string, b int) (string, string) {
2
3  }
4
5  x, y := Xpto("oi", 10)
```

Perceba que nesse caso, essa função está retornando dois valores, respectivamente para as variáveis "x" e "y".

Variadic functions

Um recurso extremamente interessante nas funções é a possibilidade de você poder informar uma quantidade indefinida de parâmetros de entrada.

Exemplo:

```
1  func variadicFunc(x ...int) int {
2
3  }
```

Nesse caso, é possível passar diversos números inteiros.

```
1  variadicFunc(1,2,5,6,19,4,5)
```

Para que você possa percorrer todos os valores, basta executar um laço de repetição utilizando o recurso "range".

```
func variadicFunc(x ...int) int {
   var res int
   for _, v := range x {
     res += v
   }
   return res
}
```

Perceba que na função acima, percorremos todos os valores de “x” e e incrementamos sua soma em uma variavel “res”.

Repare que no laço de repetição, possuimos um blank identifier seguido da variável “v”. Isso significa que a função “range” está retornando dois valores; um índice e um valor. Nesse caso, estamos preocupados apenas com o valor “v”, logo, estamos ignorando o índice através de um blank identifier.

Funções anônimas

A Go Lang também possibilita a criação de funções anônimas.

Exemplo:

```
a:= func() int {
   z +=2
}
```

ou apenas mesmo, uma função dentro de outra função:

```
func funcInsideFunc() func() int {
   x := 10
   return func() int {
     return x * x
   }
}
```

Arrays

Arrays na Go Lang nos ajuda a armazenar dados em um mesmo objeto, porém de forma fixa e totalmente tipada.

Exemplo:

```
var x [10]int
x[0] = 1
x[1] = 38
```

Uma outra forma de inicializarmos um array:

```
x := [5]int{3,4,5,6,7}
```

Slices

Podemos dizer que Slices são arrays com esteróides. Ele não possui tamanho fixo e trabalha de forma mais dinâmica, além de possuir diversos recursos.

Criando um slice:

```
slice := make([]int)
```

Por outro lado, podemos criar um slice com uma quantidade definida de posições, todavia, isso não significa que esse tamanho ficará fixado para sempre.

```
slice := make([]int, 5) // 5 posições padrão.
slice[0] = 10
fmt.Println(slice[0]) // 10
slice = append(slice, 1,2,3,47,5)
```

Uma outra forma de você declarar facilmente um slice é através do exemplo abaixo:

```
sliceString := []string {
  "Hello",
  "World"
}

fmt.Println(sliceString[0]) // "Hello"
```

Um grande recurso que slices disponibilizam na Go Lang é a possibilidade da navegação pelos índices.

Exemplo:

```go
sliceString := []string {
  "Hello",
  "World",
  "Much",
  "Better"
}

fmt.Println(sliceString[:2]) // "Hello World"
fmt.Println(sliceString[1:2]) // "World" - A partir do índice um até a segunda posiç\
ão.
fmt.Println(sliceString[2:4]) // "Much Better" - A partir do índice 2 até a quarta p\
osição.
fmt.Println(sliceString[2:]) // "Much Better" - A partir do índice 2 até ao final.
```

Maps

Os maps trabalham de forma similares com arrays associativos no PHP ou mesmo dicionários no Python. Basicamente o princípio de chave-valor.

Exemplo:

```go
m := make(map[string]int)
m["a"] = 19
m["b"] = 20
```

Você pode deletar um valor de um map:

```go
delete(m,"b")
```

Nesse ponto o valor foi apagado, porém se chamarmos o mapa no índice "b", o valor será igual a zero, uma vez que o map é de inteiros.

Para verificar se um valor existe dentro de um map, basta utilizá-lo normalmente, porém, adicione mais uma variável na chamada.

```go
_, exists := m["b"]
fmt.Println(exists) // false
```

Como não estamos interessados no valor, mas sim na existência do índice, utilizamos um blank identifier, porém, a variável exists terá um valor true ou false atribuída a ela.

Uma outra forma de definirmos um map é:

```go
var x = map[string]int{}
// ou
x := map[string]int{"a":5,"b":10}
```

Orientação a objetos na Go Lang

A Go Lango utiliza o paradigma de orientação a objetos de forma diferenciada de outras linguagens tradicionais.

Os seus "tipos" e "classes" são substituidos por estruturas de dados e tais estruturas podem ter funções atreladas diretamente a elas.

Structs

Entendimento básico sobre Structs

A Go Lang permite que criemos tipos específicos de dados.

Exemplo:

```
type CarName string
type CarYear int
type string
```

Apesar de termos os tipos declarados, não podemos relacioná-los e para isso, faremos a utilização de um tipo chamado de Struct.

```
type Car struct {
  Name string
  Year int
  Color string
}

car1 := Car{"Sedona", 2021, "Branco"}
car2 := Car{"BMW 320i", 2021, "Preto"}

fmt.Println(car1.Name)
fmt.Println(car2.Color)
```

Atachando funções em uma Struct

Para que possamos realizar ações em uma struct, é necessário que possamos atachar funções nas mesmas.

Exemplo:

```go
type Car struct {
  Name string
  Year int
  Color string
}

func (c Car) info() string {
        return c.Name + " - " + c.Color
}
```

Trabalhando com herança

Para evitar a grande repetição no processo de criação de structs, as mesmas foram pensadas para se relacionar ou "herdar" umas com as outras.

Exemplo:

```go
type CarroEsportivo struct {
  Esportivo bool
}

type Carro struct {
  Name string
  Year int
  Color string
  CarroEsportivo
}

car := Carro{"BMW 320i", 2021, "Preto", CarroEsportivo{Esportivo:true}}
```

Apesar de que na declaração tivemos que informar a Struct do CarroEsportivo, podemos chamar a propriedade herdada diretamente.

```go
fmt.Println(car.Esportivo)
// ou
fmt.Println(car.CarroEsportivo.Esportivo)
```

Serializando dados em json

A Go Lang possui recursos nativos em sua biblioteca padrão para nos ajudar serializar dados em diversos formatos, incluindo json.

Exemplo:

```go
type Carro struct {
  Name string
  Year int
  Color string
}

car := Carro{"BMW 320i", 2021, "Preto"}
result, _ := json.Marshal(car) // dados estão em bytes
fmt.Println(string(result)) //transformando dados em string
```

Mesmo no processo de serialização, a regra de visibilidade é aplicada. Logo, os atributos de uma struct para serem serializados deverão estar com o primeiro caractere em maiúsculo.

Trabalhando com tags

Tag é um mecanismo da linguagem para conseguir marcar um atributo de uma struct. Essa marcação pode ter diversos objetivos. No exemplo que apresentaremos agora, será para facilitar o processo de manipulação da serialização dos dados para json.

```go
type Carro struct {
  Name string
  Year int `json:"-"`
  Color string
}
```

Perceba que no exemplo acima, no atributo Year, adicionamos `json:"-"`. Isso significa que quando os dados forem serializados para json, o atribudo Year não deverá ser exibido.

Além de termos a opção de ocultar um atributo durante o processo de serialização, podemos também alterar a chave do atributo:

```go
type Carro struct {
  Name string `json:"modelo"`
  Year int `json:"-"`
  Color string
}
```

No exemplo acima estamos informando que quando o atributo “Name” for serializado, a chave que deve ser exibida no json será “modelo” ao invés de “Name”.

Realizando o bind de um Json para uma Struct

Muitas vezes recebemos um json e queremos fazer o bind em uma struct.

```go
type Car struct {
  Name string
  Year int
  Color string
}

var car Car
json := []byte(`{"Name":"BMW 320","Year":2021,"Color":"Black"}`)

json.Unmarshal(json, car)
fmt.Println(car.Name) // BMW 320
```

Interfaces

A Linguagem Go também possui o conceito de Interfaces, ou seja, é um tipo de objeto que possui uma definição.

Quando uma struct possui a exata definição de tal interface, isso significa que a struct está a implementando, logo, ela também pode ser considerada do mesmo tipo que a interface.

Diferentemente de linguagens tradicionais orientadas a objetos, a Go Lang não obriga de forma declarativa que uma struct tenha que implementar uma interface usando instruções como “implements”. A implementação se da de forma automática. Basta a struct possuir a mesma assinatura informada na interface.

No exemplo abaixo criaremos um struct e uma interface que possuem a mesma assinatura.

```go
type vehicle interface {
  start() string
}

type car struct {
  Name string
}

func (c car) start() string {
 return "O carro iniciou"
}
```

Perceba acima que mesmo a struct “car” não informando de forma explícita, a mesma está implementando a interface vehicle. Nesse ponto podemos trabalhar com algo do tipo:

```
1 func BrincadeiraComCarro(v vechicle) {
2   return v.start()
3 }
4
5 var c car
6
7 BrincadeiraComCarro(c)
```

Perceba que a struct do tipo car pode ser interpretada como “vehicle” uma vez que ela implementa tal interface.