

INTRODUÇÃO À LINGUAGEM PET - ADS



À

= GO

Slides
complementares e
completos para
estudiosos
que gostariam de
estudar mais sobre
GO

PET - ADS → Responsáveis por esse tutorial



Criado pela CAPES, o programa de educação tutorial visa a formação completa dos alunos de graduação bem como fomenta as pesquisas em universidades Brasil à fora

Para saber mais sobre o PET do curso de ADS (IFSP campus São Carlos) acesse:

<https://www.facebook.com/PETADSIFSP/>

http://pet_ads.pginas.scl.ifsp.edu.br/src/paes/home/index.html

O QUE
VEREMOS
AQUI?

INTRODUÇÃO

AMB. DE PROGRAMAÇÃO

OPERADORES, VARIÁVEIS E PACOTES

ESTRUTURAS CONDICIONAIS

ESTRUTURAS DE REPETIÇÃO

ARRAYS E SLICES

MAPS

APRENDIZAGEM RÁPIDA

É fácil para o iniciante e para o avançado

LINGUAGEM PRAGMÁTICA

É fácil entender o que o outro fez, logo propicia desenvolvimento em grandes equipes

ADAPTADA

GOLANG tem sido popularizada em novos ambientes, como a programação para a nuvem

FORTEMENTE TIPADA

Aparenta ser fracamente tipada, o que facilita as implementações

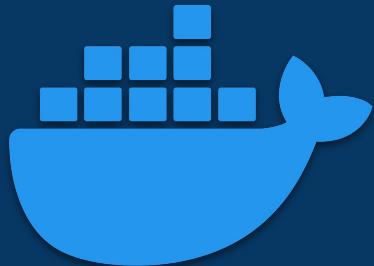
GARBAGE COLLECTOR

GO dispensa gerenciamento de memória manual

RÁPIDA

É uma linguagem tão ágil como Python e vem crescendo em uso nas áreas de ciência de dados

POR QUE = GO ?



docker®

*IMPLEMENTAÇÕES COM A
GOLANG*



kubernetes



CLOUD FOUNDRY



TensorFlow



Veja mais em:

<https://blog.mandic.com.br/artigos/porque-a-linguagem-go-e-a-mais-popular-de-todos-os-tempo/>

SURGIMENTO DA LINGUAGEM GO



NECESSIDADE

- Linguagem performática
- Eficiência
- Fácil compilação

Qual seria a
melhor
linguagem?



NENHUMA?!?!?!

Vamos criar a nossa =)

CRIADORES



KEN THOMPSON, ROB PIKE E ROBERT GRIESMER

GO, go, golang, GOLANG, GoLang são sinônimos ;)

ANTES DA PRÁTICA

Nós já vamos para a parte legal, calma ai, vou te dar umas dicas antes

CONTEÚDO EXTRA

Nesse minicurso, os programas serão feitos em uma IDE online.

Para mais informações sobre Go e também sobre IDEs, acesse os links a seguir:

GO POR EXEMPLO

<http://goporexemplo.golangbr.org/>

APRENDA GOLANG

<https://aprendagolang.com.br/>

GOLANG TOUR

<https://go.dev/tour/welcome/1>

AWESOME GO

<https://github.com/avelino/awesome-go>

SIGA O TUTORIAL OU
CLIQUE NO LINK!

VAMOS COMEÇAR!

<https://go.dev/play/>

Google

Pesquise no Google ou digite um URL



GOLANG



<https://go.dev> Traduzir esta página

The Go Programming Language

Build fast, reliable, and efficient software · Go is an open source programming language supported by Google · Easy to learn, efficient, and get started with · Built-in ...

Downloads

After downloading a binary release



Install Go on your system, please ...

Get Started

Install the latest version of Go. For instructions to download and ...

Documentation

Tutorial: Get started - Download and install - How to Write Go Code

Tutorial: Get started

Install Go (if you haven't already). · Write some simple "Hello ...

[Mais resultados de go.dev »](#)

As pessoas também perguntam



Why Go

Get Started

Docs

Packages

Play

Blog

Build fast, reliable, and efficient software at scale

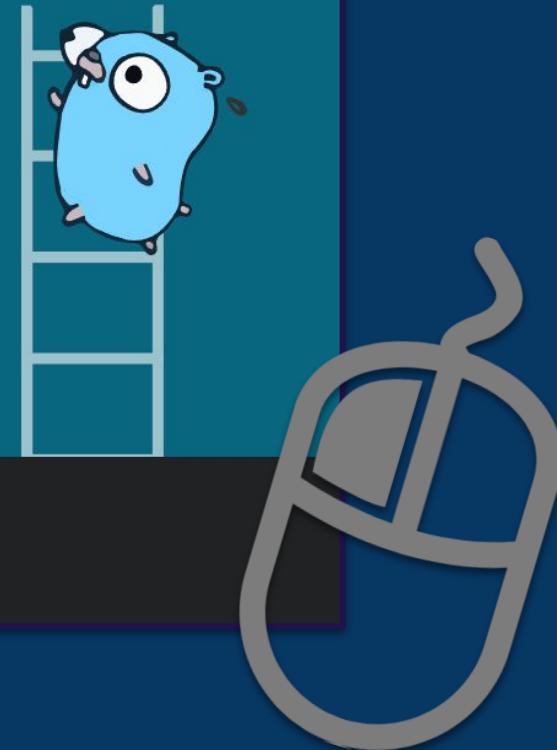
- ✓ Go is an open source programming language supported by Google
- ✓ Easy to learn and get started with
- ✓ Built-in concurrency and a robust standard library
- ✓ Growing ecosystem of partners, communities, and tools

Get Started

Download

Download packages for [Windows 64-bit](#), [Linux](#), and more

The go command by default downloads and updates modules using the Go module mirror and Go checksum database run by Google. [Learn more.](#)



Companies using Go

Organizations in every industry use Go to power their software and services [View all stories](#)

Install the latest version of Go

Install the latest version of Go. For instructions to download and install the Go compilers, tools, and libraries, [view the install documentation](#).

[Download](#)

Download packages for [Windows 64-bit](#), [macOS](#), [Linux](#), and [more](#).

Documentation

Everything there is to know about Go. Get started on a new project or brush up for your existing Go code.

[View documentation](#) 

Tour of Go

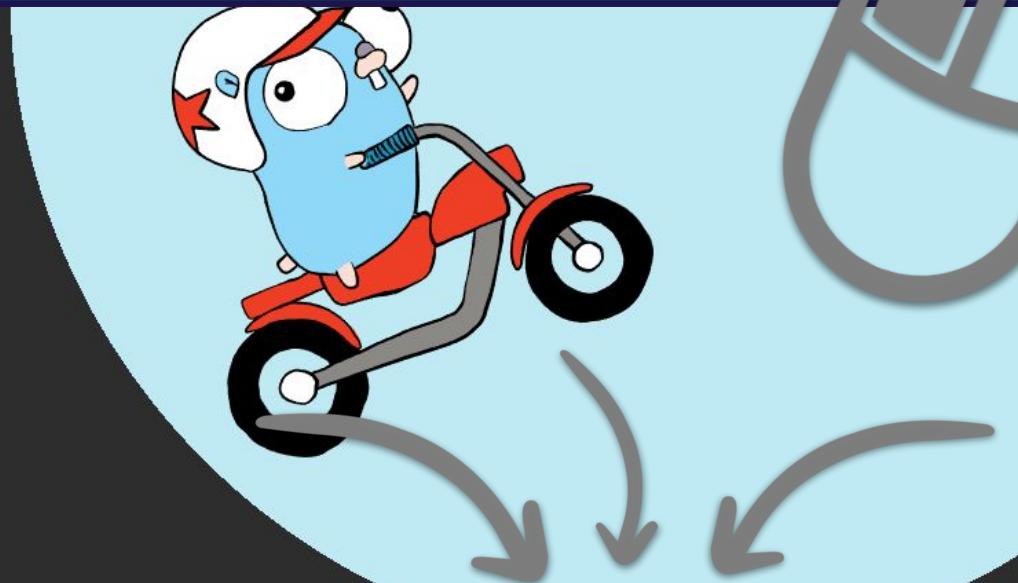
An interactive introduction to Go in three sections. Each section concludes with a few exercises so you can practice what you've learned.

[Take a tour](#) 

Playground

Playground allows anyone with a web browser to write Go code that we immediately compile, link, and run on our servers.

[Go to playground](#) 



The Go Playground

```
1 // You can edit this code!
2 // Click here and start typing.
3 package main
4
5 import "fmt"
6
7 func main() {
8     fmt.Println("Hello, 世界")
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

VERSÃO DA
LINGUAGEM

Go 1.19

Run

Format

Share

Hello, World!

NÃO MEXER AQUI!

AMBIENTE DE PROGRAMAÇÃO

EXEMPLOS
DE CÓDIGOS,
MAS HOJE
NÓS VAMOS
CODAR
NOSSOS
PRÓPRIOS
EXEMPLOS
=)

ESTRUTURA DO NOSO CÓDIGO



```
package main  
  
//pacotes e bibliotecas  
import "..."  
  
func main(){  
    //programa principal  
}
```

COMENTÁRIOS



Run

Format

Share

COMPILA
E RODA

FORMATÁ
A INDENTAÇÃO

CRIA UM LINK
COMPARTILHÁVEL

NESSA LINGUAGEM EXISTEM
DUAS MANEIRAS DE SE
DECLARAR UMA VARIÁVEL

COM A LONGA DECLARAÇÃO,
OU A CURTA. VAMOS
UTILIZAR NA PRIMEIRA PARTE
DA AULA A CURTA, LOGO:

`:=` É DIFERENTE DE `=`

`:=`
DECLARA A
VARIÁVEL E
ATRIBUI VALOR

`=`
APENAS
ATRIBUI VALOR



TOME NOTA!



OPERADORES, VARIÁVEIS & PACOTE FMT



Luana
Monteiro

COMPARATIVOS

Comparam duas coisas →
ex: 2 é maior que 4?

LÓGICOS

Fazem operações lógicas
entre duas coisas → and, or

ARITMÉTICOS

Fazem contas → ex: $2+2 = 4$

OPERADORES

ATRIBUTIVOS

Atribuem um valor a uma variável
→ ex: quatro := 4

ARITMÉTICOS

+

fmt.Println(num + numD)

SOMA

6

-

fmt.Println(num - numD)

SUBTRAÇÃO

2

*

fmt.Println(num * numD)

MULTIPLICAÇÃO

8

fmt.Println(num / numD)

DIVISÃO

2

package main

import "fmt"

func main(){

num := 4

numD := 2

fmt.Println(num "operador aritmético" numD)

}

ARITMÉTICOS

%

```
fmt.Println(num % numD)
```

RESTO

0

++

```
num++  
numD++  
fmt.Println(num, numD)
```

INCREMENTO

5 3

```
package main
```

```
import "fmt"
```

```
func main(){  
    num := 4  
    numD := 2  
    fmt.Println(num "operador aritmético" numD)  
}
```

```
num--  
numD--  
fmt.Println(num, numD)
```

DECREMENTO

3 1

fmt.Println(num > numD)

MAIOR QUE

true

fmt.Println(num < numD)

MENOR QUE

false

fmt.Println(num >= numD)

MAIOR OU
IGUAL À

true

fmt.Println(num <= numD)

MENOR OU
IGUAL À

false

package main

import "fmt"

```
func main(){  
    num := 4  
    numD := 2  
    fmt.Println("operador comparativo", numD)}
```

}

COMPARATIVOS

fmt.Println(num == numD)

IGUAL?

false

fmt.Println(num != numD)

DIFERENTE?

true

package main

import "fmt"

func main(){

 num := 4

 numD := 2

 fmt.Println(num "operador comparativo" numD)

}

COMPARATIVOS

&&

```
criterio := num == 5 && numD == 8
```

false

AND

true

*NEGAÇÃO: USADO JUNTO COM O “=” PARA ANALISAR
RESULTADOS LOGICAMENTE DIFERENTES*

```
criterio := num == 4 || numD == 8
```

false

true

```
criterio := num == 4 || numD == 2
```

```
criterio := num == 12 || numD == 8
```

```
criterio := num == 12 || numD == 8
```

package main

import "fmt"

func main(){

num := 4

numD := 2

criterio := num == 4 “operador logico” numD == 2

fmt.Println(criterio)

}

LÓGICOS

`num = 9`

ATRIBUI

`9`

```
package main  
  
import "fmt"  
  
func main(){  
    num := 4  
    num "operador atributivo" 9  
    fmt.Println(num)  
}
```

`num -= 9`

*SUBTRAIE
ATRIBUI*

`-5`

`num += 9`

*SOMA E
ATRIBUI*

`13`

ATRIBUTIVOS

num /= 9

DIVIDE E
ATRIBUI

0

```
package main

import "fmt"

func main(){
    num := 4
    num "operador atributivo" 9
    fmt.Println(num)
}
```

num %= 9

ENCONTRA
O RESTO E
ATRIBUI

4

%=

num *= 9

MULTIPLICA
E ATRIBUI

36

*=

ATRIBUTIVOS



HORA DO
DESAFIO!

o que será impresso na saída da execução?

```
num := 80  
num %= 9  
fmt.Println(num)
```

A

```
num :=17  
numD := 121  
crit := num == 10 || numD > 54  
fmt.Println(crit)
```

C

```
x := 52  
fx := (x + 8) * 2  
crit := x <= 21 && fx >= 90  
fmt.Println(crit)
```

B

```
num := 180  
numD := 14  
crit := num == 180 && numD <= 26  
fmt.Println(crit)
```

D

GABARITO

A. 8

B. FALSE

C. TRUE

D. TRUE

num := 14
numD := 9
fmt.Println(numD * num)

A

num := 38
num -= 12
fmt.Println(num)

B

num := 3
num *= 15
fmt.Println(num)

C

x := 52
fx := (x + 548) * 2
x += 70
fx *= fx * 2
fmt.Println(x, fx)

F

num := 154
numD := 9
fmt.Println(numD > num)

G

num := 10
numD := 99
fmt.Println(numD >= num)

J

D

num := 21
num %= 9
fmt.Println(num)

N

num := 80
num %= 9
fmt.Println(num)

E

num := 3
numD := 15
fmt.Println(numD == num)

K

x := 52
fx := (x + 8) * 2
crit := x <= 21 && fx >= 90
fmt.Println(crit)

I

num := 17
numD := 121
crit := num == 10 || numD > 54
fmt.Println(crit)

L

num := 32
numD := 15
fmt.Println(numD <= num)

M

num := 180
numD := 14
crit := num == 180 && numD <= 26
fmt.Println(crit)

O

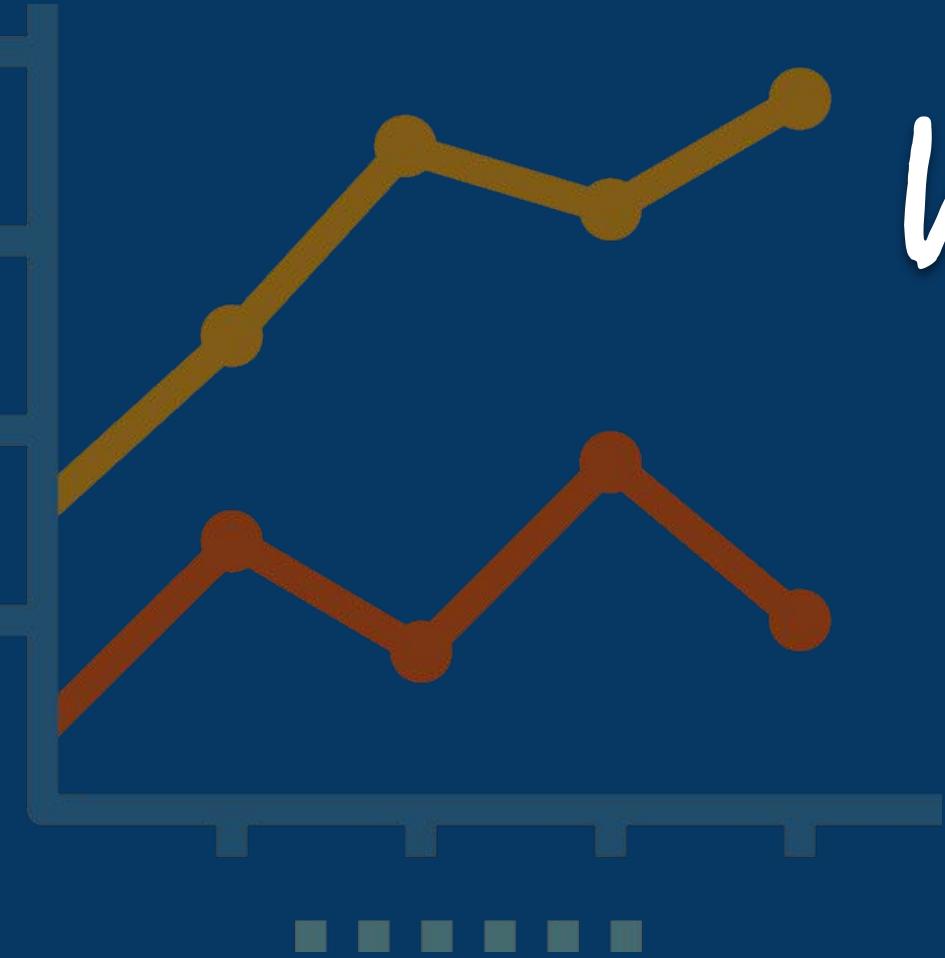
x := 477
x %= 6
fx := (x + 13) * 2
fmt.Println(x, fx)

P

GABARITO

- A. 126
- B. 26
- C. 45
- D. 108
- E. 8
- F. 122 2880000
- G. FALSE
- H. 120
- I. FALSE
- J. TRUE
- K. FALSE
- L. TRUE
- M. TRUE
- N. 3
- O. TRUE
- P. 32

VARIÁVEIS



O QUE SÃO?

Variáveis são como caixas de tamanhos específicos próprias para um tipo de item

Por exemplo: uma caixa de cupcakes só serve para cupcakes



Logo uma “caixa” para números inteiros só serve para números inteiros, e assim por diante.

Na tabela do próximo slide podemos observar quais os tipos de variáveis existem na golang e para quais tipos de números elas servem

TIPOS IMPORTANTES

TIPO	FAIXA DE VALORES	PRECISÃO	TAMANHO
INT32	-2.147.483.648 a 2.147.483.647	-	32 BITS
INT64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	-	64 BITS
FLOAT32	$\pm 1,5 \times 10^{-45}$ para $\pm 3,4 \times 10^{38}$	6 a 9 DÍGITOS	32 BITS
FLOAT64	$\pm 5,0 \times 10^{-324}$ para $\pm 1,7 \times 10^{308}$	15 A 17 DÍGITOS	64 BITS
BOOL	DOIS - TRUE OU FALSE	-	1 BIT
STRING	TODO TIPO TEXTUAL	-	8 BITS P/ CARACTER

LEIA MAIS SOBRE AS VARIAVEIS DA GOLANG EM:

<https://www.treinaweb.com.br/blog/conhecendo-variaveis-e-constantes-na-golang>

XOXO
!=
xoxo
!=
XoxO

só um tipo
de dado

REGRINHAS

declare o
tipo!

FLOAT32
INT VAR BOOL
STRING

nomes iniciam-se
com letras,
podendo conter
números

atenção ao
escopo!!!

global?
fora dos
blocos

local?
dentro dos
blocos

var nome tipo = valor (opcional)

nome := valor

:= → operador curto de
declaração

```
package main

//pacotes e bibliotecas
import “...”

//var nome tipo = valor
var inteiro int = 2022
var flutuante float32
var texto string = “PET”
var booleano bool

func main(){
    //programa principal
}
```

declaração completa fora
dos blocos de função

Exemplificando

```
package main

//pacotes e bibliotecas
import “...”

func main(){
    //nome := valor
    inteiro := 2022
    flutuante := 20.21
    texto := “ADS”
    booleano := true
}
```

op. de curta declaração

é possível
declarar
múltiplas
variáveis
em
conjunto

```
//var(  
//    nome tipo = valor  
//    nome tipo  
//)  
  
var(  
    inteiro int = 2022  
    flutuante float32  
    texto string = “pet ads”  
    booleano bool  
)  
  
func main(){  
    //nome, nome, nome, ... := valor, valor, valor, ...  
    inteiro, flutuante, texto, booleano := 2022, 20.21, “ADS”, true  
}
```

Não façam
isso! não é
uma boa
prática

INT → *STRING*

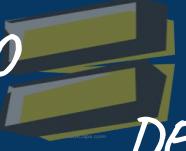
```
func main(){  
    flutuante := 20.22  
    //conversão:  
    //nome := tipo(valor)  
    inteiro := int(flutuante)  
}
```

CONVERTENDO VARIÁVEIS

APESAR DE NÃO APARENTAR,
A LINGUAGEM É
FORTEMENTE TIPADA COM
VARIÁVEIS ESTÁTICAS, LOGO
PRECISAMOS CONVERTER
UM VALOR EM OUTRO EM
DETERMINADOS MOMENTOS

SUBLINHADO → *DADO INÚTIL PODENDO
SER DESCARTADO*

CRIANDO O PRÓPRIO TIPO

*NOME
ESCOLHIDO*  *TIPO
DETERMINADO*

```
//type nome tipo  
type PET int
```

```
var numero PET = 2022
```

O TIPO É UMA FERRAMENTA ÚTIL PARA MUITAS OUTRAS COISAS COMO PODER ALTERAR O TIPO DE UMA VARIÁVEL DE MANEIRA RÁPIDA (COMO CONSTANTES)

O TIPO TAMBÉM É USADO PARA DEFINIR STRUCTS (ESTRUTURAS COM VÁRIOS TIPOS DE VARIÁVEIS DENTRO) E DETERMINAR INTERFACES (BLOCOS DE CÓDIGOS COMPORTAMENTAIS PARA AS STRUCTS)

PACOTES



FMT →

ENTRADA E SAÍDA DE DADOS

```
package main  
  
import "fmt"  
  
func main(){  
    //programa principal  
}
```

FMT = format
ForMaT

COMANDO	FUNÇÃO
fmt.Println(parâmetros)	Print dos parâmetros com \n no final
fmt.Print(parâmetros)	Print dos parâmetros sem \n no final
fmt.Sprint(parâmetros)	Retorna a concatenação dos parâmetros
fmt.Printf(parâmetros)	Com %v mostra o valor da variável Com %T mostra o tipo da variável

SAÍDA DE DADOS

```
package main

import "fmt"

func main(){
    var x int = 20
    var y float32 = 33.53
    var z string = "PET"
    fmt.Println("o próximo vem aqui na mesma linha ")
    fmt.Println("eu estou na mesma linha, mas o próximo vai na de baixo")
    fmt.Printf("eu uso o símbolo de percentual para imprimir isso x = %d ", x)
    fmt.Printf("y = %f z = %s", y, z)
}
```

o próximo comando vem aqui na mesma linha
eu estou na mesma linha, mas o próximo vai
na de baixo

eu uso o símbolo de percentual para imprimir
isso 20 33.53 PET

SAÍDA DE DADOS

```
package main

import "fmt"

func main(){
    var x int = 12
    var z string = "PET"
    fmt.Println("O Sprint concatena valores de diferentes tipos")
    fmt.Println("Afinal isso não é possível na golang")
    fmt.Println("Um exemplo:")
    // resposta := "O " + z + " tem " + x + " bolsistas" //erro! não posso juntar string e int
    resposta := fmt.Sprintf("O %s tem %d bolsistas", z, x)
    fmt.Printf(resposta)
}
```

O PET tem 12 bolsistas

ENTRADA DE DADOS

COMANDO	FUNÇÃO
fmt.Scanln(parâmetros)	captura a linha toda, se não tiver nada a ser capturado um enter faz seguir o código
fmt.Scan(parâmetros)	captura a linha toda, mas não segue o código enquanto não capturar algo
fmt.Scanf(parâmetros)	dá para encaixar um \n no fim, mas não funciona em windows >:

ENTRADA DE DADOS

```
package main

import "fmt"

func main() {
    var nome, curso, nome_prof string
    var idade int
    var ira float32
    fmt.Println("Informe seu nome e sua idade:")
    fmt.Scanln(&nome, &idade)
    fmt.Print("Informe seu curso e o nome do seu professor preferido:\n")
    fmt.Scan(&curso, &nome_prof)
    fmt.Println("Digite seu IRA:")
    fmt.Scanf("%f\n", &salario)
    fmt.Printf("\nNome: %s\nIdade: %d\nCurso: %s\nNome do professor: %s\nIRA: %f", nome, idade,
    curso, nome_prof, ira)
}
```

Nome: Juscelino
Idade: 199
Curso: AAAao
Nome do professor: aaaAA
IRA: 9889.09

PERGUNTAS?



A dark, close-up portrait of Jigsaw from the movie Saw. He is wearing his signature white mask with black eye holes, red lips, and a red bow tie. His hair is dark and messy. The background is dark and moody.

EXERCÍCIOS DE NÍVEL #1

LET'S PLAY A GAME

Utilizando o operador curto de declaração, atribua estes valores às variáveis x, y e z, respectivamente:

- o 2022
- o PET
- o true

Agora, demonstre os valores nas variáveis com:

- o uma única declaração print
- o múltiplas declarações print

Crie um tipo próprio com int e - utilizando a declaração por extenso - uma variável x com este tipo.

Na função main:

- o Demonstre o valor da variável x
- o Demonstre o tipo da variável x
- o Atribua 2022 à variável x utilizando o operador de atribuição
- o Demonstre o valor da variável x

Utilizando o operador curto de declaração, atribua estes valores às variáveis x, y e z, respectivamente:

- o 2022
- o PET
- o true

Agora, demonstre os valores nas variáveis com:

- o uma única declaração print
- o múltiplas declarações print

```
package main
```

```
import "fmt"
```

```
func main(){  
    //variaveis  
    x := 2022  
    y := "PET"  
    z := true  
    // Único print  
    fmt.Println(x, y, z)
```

```
//Múltiplos prints  
fmt.Println(x)  
fmt.Println(y)  
fmt.Println(z)
```

```
}
```

Crie um tipo próprio com int e - utilizando a declaração por extenso - uma variável x com este tipo

Na função main:

- o Demonstre o valor da variável x
- o Demonstre o tipo da variável x
- o Atribua 2022 à variável x utilizando o operador de atribuição
- o Demonstre o valor da variável x

```
package main

import "fmt"
//Criação do tipo
type pet int

//declaração
var x pet

func main(){
    //demonstrando o valor e o tipo
    fmt.Printf("%v", x)
    fmt.Printf("\n%T", x)
    //atribuição
    x := 2022
    //demonstração
    fmt.Printf("\n%v", x)
}
```

ESTRUTURAS: CONDICIONAIS & DE REPETIÇÃO



IF
ELSE
ELSE IF

```
package main

import "fmt"

func main(){
    nome := "Carol"
    sobrenome := "Shaw"
    if nome == "Carol"{
        if sobrenome == "Shaw"{
            fmt.Println("Carol Shaw")
        } else {
            fmt.Println("Carol")
        }
    } else if nome == "Pedro"{
        fmt.Println("Seu nome é Pedro")
    }
}
```

O IF É UMA CONDICIONAL

É TRADUZIDO COMO
“SE”

RETORNA UM CÓDIGO PARA
UMA AVALIAÇÃO VERDADEIRA
DE UMA PROPOSIÇÃO LÓGICA

OU SEJA:

SE ESTIVER FRIO VISTA UM
CASACO

O IF VERIFICA SE ESTÁ FRIO, SE
ESTIVER ELE VESTE O CASACO

O QUE É UM IF?

VAMOS OBSERVAR ISSO
EM UM CÓDIGO EM GO

```
package main

import "fmt"

func main(){
    clima := "Frio"
    if clima == "Frio"{
        fmt.Println("vista um casaco")
    }
}
```

O ELSE É COMO UM OPOSITOR

É TRADUZIDO COMO
“SE NÃO”

RETORNA UM CÓDIGO PARA
UMA AVALIAÇÃO FALSA DE
UMA PROPOSIÇÃO LÓGICA, É
OPCIONAL

OU SEJA:

SE ESTIVER FRIO VISTA UM
CASACO, SE NÃO VISTA UMA
REGATA

O IF VERIFICA SE ESTÁ FRIO, SE
NÃO ESTIVER ELE VESTE A
REGATA

E SE ESTIVER CALOR?

```
package main

import "fmt"

func main(){
    clima := "Calor"
    if clima == "Frio"{
        fmt.Println("vista um casaco")
    } else {
        fmt.Println("vista uma regata")
    }
}
```

O QUE ESSE CÓDIGO RETORNA?

```
package main

import "fmt"

func main(){
    clima := "Ventando"
    if clima == "Frio"{
        fmt.Println("vista um casaco")
    } else {
        fmt.Println("vista uma regata")
    }
}
```

DETALHE

O ELSE É UMA RESPOSTA GENÉRICA, QUALQUER COISA QUE NÃO FOR COMPATÍVEL COM O IF ENTRA PARA O ELSE

COMO ESPECIFICAR OUTRA RESPOSTA POSSÍVEL?

- ELSE IF
- SWITCH CASE

O ELSE IF É UMA OPOSIÇÃO CRITERIOSA

ELSE IF

É TRADUZIDO COMO
“SE NÃO SE”

AVALIA COMO O IF, MAS SENDO
UMA OUTRA PROPOSIÇÃO

OU SEJA:

SE ESTIVER FRIO VISTA UM CASACO,
SE NÃO ESTIVER FRIO MAS ESTIVER
CALOR VISTA UMA REGATA

O IF VERIFICA SE ESTÁ FRIO, O ELSE
IF VERIFICA SE ESTÁ CALOR E O
ELSE DÁ UMA RESPOSTA GENÉRICA

```
package main

import "fmt"

func main(){
    clima := "Calor"
    if clima == "Frio"{
        fmt.Println("vista um casaco")
    } else if clima == "Calor"{
        fmt.Println("vista uma regata")
    } else{
        fmt.Println("vista o que quiser")
    }
}
```

package main

import “fmt”

```
func main(){
    autor := “Orwell”
    switch autor{
        case “Lovecraft”:
            fmt.Println(“H. P. Lovecraft”)
        case “Poe”:
            fmt.Println(“Edgar Allan Poe”)
        default:
            fmt.Println(“O autor não está na biblioteca”)
    }
}
```

SWITCH CASE

COMO FUNCIONA O SWITCH?

É COMO UM IF QUE AVALIA O ESTADO DE UMA ÚNICA VARIÁVEL

É COMPOSTO POR:

FORMAS PREVISTAS QUE A VARIÁVEL PODE ASSUMIR

UM CASO PADRÃO, PARA QUANDO NENHUMA DAS FORMAS PREVISTAS FORAM COMPATÍVEIS COM O VALOR ASSUMIDO

O BLOCO DE CÓDIGOS A SER EXECUTADO PARA CADA CASO

UM EXEMPLO EM GO

- A VARIÁVEL ASSUME O NOME DE UM LIVRO PROCURADO EM UM ACERVO
- CADA LIVRO PODE TER DOIS ESTADOS DISPONÍVEL OU EMPRESTADO
- SE O LIVRO FOR EMPRESTADO VAMOS MOSTRAR O NOME DE QUEM O EMPRESTOU

```
package main
```

```
import "fmt"
```

```
func main(){
    var estado string;
    livro := "Clean Code"
    switch livro{
        case "Pragmatic programmer":
            estado = "emprestado"
            if estado == "emprestado"{
                fmt.Println("emprestado para Felipe")
            } else{fmt.Println("disponivel")}
        case "Clean Code":
            estado = "emprestado"
            if estado == "emprestado"{
                fmt.Println("emprestado para Bianca")
            } else{fmt.Println("disponivel")}
        default:
            fmt.Println("Não temos o livro")
    }
}
```

emprestado para Bianca

DESAFIO

- Se alterarmos o estado do livro para "disponível", o que o programa responderá?
- E se alterarmos o livro para "Pragmatic Programer"?
- O que acontece se colocarmos um título que não está dentro dos casos do switch?



FOR

NA GOLANG NÃO EXISTE OUTRO TIPO DE LAÇO DE REPETIÇÃO QUE NÃO SEJA O FOR, OUTRAS ESTRUTURAS COMO O WHILE NÃO SÃO COMPREENDIDAS POR ELA

CLÁSSICA
HIERÁRQUICA
CONDICIONAL
INFINITA



WHILE

```
package main
```

```
import "fmt"
```

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

CLÁSSICO

FOR COMUM, COM UM
INICIALIZADOR, UMA
CONDIÇÃO DE PARADA E
EXECUÇÃO

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

UM LOOP DENTRO DE OUTRO

HIERÁRQUICA

```
package main
```

```
import "fmt"
```

```
func main(){
    for horas:= 0; horas <= 2; horas++{
        fmt.Println("Hora:", horas)
        for x:= 0; x <= 4; x++{
            fmt.Println(x)
        }
        fmt.Print("\n")
    }
}
```

Hora 0 :
0 1 2 3 4
Hora 1:
0 1 2 3 4

```
package main

import "fmt"

func main(){
    x := 0
    for x < 5{
        x++
        fmt.Println(x)
    }
}
```

EQUIVALE AO WHILE

AVALIA SE UMA
CONDIÇÃO É VERDADEIRA
PARA UMA VARIÁVEL
MODIFICADA DENTRO DO
LOOP

- 1 TRADUZINDO PARA O EXEMPLO:
- 2 PARA X<5, INCREMENTO O
- 3 VALOR E IMPRIME NA TELA,
- 4 QUANDO X FOR 5 O FOR ACABA
- 5

CONDICIONAL

INFINITA → *BREAK*

NADA ESPECIFICADO PARA O LAÇO, ASSIM ELE CONTINUA ATÉ ENCONTRAR ALGO QUE O PARE, UM BREAK POR EXEMPLO

```
package main
import "fmt"

func main(){
    for{
        fmt.Println("ao infinito e além")
        break
    }
}
```

CONTINUE

O OPOSTO DO BREAK,
CONTINUE PODE SER
APLICADO PARA
QUALQUER FOR, ELE
SERVE PARA SITUAÇÕES
ONDE SE QUER PULAR
UM RESULTADO POR
EXEMPLO

```
package main
import "fmt"

func main(){
    for i := 0; i < 5; i++{
        if i % 2 != 0{continue}
        fmt.Println(i)
    }
}
```

0
2
4

Esse código imprime todos os i's com exceção aos ímpares, pois o código encontra um continue, que faz ele pular a última sentença e repetir o laço



DÚVIDAS?

MARIO
001600

0 0003

WORLD
1-1

TIME
335

NEXT LEVEL!

EXERCÍCIOS DE NÍVEL #2



Utilizando o laço de repetição for, exiba os números de 1 a 100.

Utilizando a repetição condicional, crie um loop que conte os anos de 2016 até 2022.

Agora, utilizando a repetição infinita e estruturas condicionais, crie um loop que conte os anos de 2002 até 2022.

Demonstre o resto da divisão por 3, de todos números entre 9 e 100.

Utilizando o laço de repetição for, exiba os números de 1 a 100.

```
package main
import "fmt"

func main(){
    for i := 1; i < 100; i++{
        fmt.Println(i)
    }
}
```

Utilizando a repetição condicional, crie um loop que conte os anos de 2016 até 2022.

```
package main
import "fmt"

func main(){
    anoIni := 2016
    anoFim := 2022
    for anoIni <= anoFim{
        fmt.Println(anoIni)
        anoIni++
    }
}
```

Agora, utilizando a repetição infinita e estruturas condicionais, crie um loop que conte os anos de 2002 até 2022.

```
package main
import "fmt"

func main(){
    anoIni := 2002
    anoFim := 2022
    for{
        if anoIni > anoFim{
            break
        }
        fmt.Println(anoIni)
        anoIni++
    }
}
```

Demonstre o resto da divisão por 3, de todos números entre 9 e 100.

```
package main
import "fmt"

func main(){
    for i:=9; i <= 100; i++{
        fmt.Println(i % 3)
    }
}
```

AGRUPAMENTO DE DADOS

=GO

SLICES ARRAYS MAPS



ARRAYS

- Limitadas
- Indexadas
- Só um tipo
- itens vazios recebem 0



POUCO UTILIZADAS NA LINGUAGEM GO

```
//declarações globais
//array com 10 posições vazio
var arrayGlobalVazio[10]int

//array de duas posições com conteúdo
var arrayGlobalComConteudo = [2]bool{true, false}

func main(){
    //arrays de declarações locais (na função que os usa)
    arrayLocalVazio := [10]string{ }

    arrayLocalComConteudo := [2]float32{20.10, 23.22}

}
```

DECLARAÇÃO

OPERAÇÕES LÓGICAS:

- Encontrar um item
- Atualizar o item
- Adicionar um item
- Remover um item

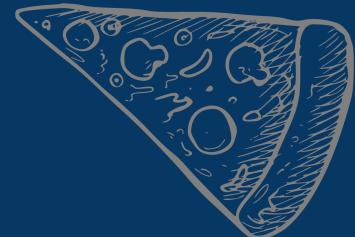
“ARRAYS ILIMITADAS”

SLICES

INICIALIZADA COMO
ARRAY SEM
QUANTIDADE

TAMBÉM PODE SER
INICIALIZADA COM A FUNÇÃO
MAKE

APPEND ADICIONA



```
//declarações globais
//array com 10 posições vazio
var SliceGlobalVazio[10]int

//array de duas posições com conteúdo
var SliceGlobalComConteudo = [2]bool{true, false}

func main(){
    //arrays de declarações locais (na função que os usa)
    SliceLocalVazio := [10]string{}

    SliceLocalComConteudo := [2]float32{20.10, 23.22}

}
```



UMA SLICE É
UMA ARRAY
QUE AINDA
NÃO FOI
ALTERADA
NO CÓDIGO!

```
//declarações globais
//var nome = make(tipo, quantidade, capacidade)
var SliceGlobal = make([]int, 1, 2)

func main(){
    //arrays de declarações locais (na função que os usa)
    SliceLocal := make([]string, 3, 4)

}
```

USANDO A FUNÇÃO MAKE PARA FAZER UM SLICE

ALGUMAS UTILIDADES PARA SLICES

COMANDO	FUNÇÃO
nome[índice]	retorna elemento no índice indicado
nome[índice] = valor	altera o elemento do índice indicado
len(nome)	retorna o comprimento da slice
cap(nome)	retorna a capacidade da slice
nome = append(nome, elemento 1...)	adiciona os elementos no fim da slice

PERCORRER = FOR
FATIAR = SLICE



UMA VEZ FATIADO A SLICE ANTERIOR E INUTILIZADA, POIS FOI ALTERADA

fatia := slice[:atéAqui]
fatia := slice[apartirDaqui:]

EXCLUINDO E ANEXANDO ITENS

```
slice := [] int{1, 2, 3}
```

```
novo := append(slice[:1], slice[2:]...)
```

PARA REMOVER UM ITEM DE UM SLICE
É PRECISO ADICIONAR O INTERVALO
CORRESPONDENTE A AUSÊNCIA DESSE
ITEM EM OUTRO SLICE

OU SOBRESCREVER O ITEM QUE DESEJA
APAGAR COM O RESTANTE DOS ITENS NO
SLICE, OU SEJA FAZER UM FOR ONDE:
 $ITEM[I] = ITEM[I+1]$

APPEND

EXCLUINDO E ANEXANDO ITENS

PARA UNIR OS DOIS SLICES APENAS
USA-SE O APPEND PARA AMBOS
OU USA-SE UM FOR PERCORRENDO
AMBOS E COPIANDO O CONTEÚDO DE
SLICE1 PARA NOVO E DEPOIS SLICE2
PARA NOVO

APPEND

```
slice1 := [] int{1, 2, 3}  
slice2 := []int{4, 5, 6}
```

```
novo := append(slice1, slice2...)
```

ISSO CONFIGURA UMA “LISTA DE
LISTAS”
OU UMA MATRIZ
EM GO TAMBÉM CONHECIDO COMO
SLICES MULTIDIMENSIONAIS

SLICES MULTIDIMENSIONAIS

IDEIA PARECIDA COM MATRIZES

UM SLICE RECEBE OUTROS N SLICES COM A MESMA QUANTIDADE DE ITENS E COM AS MESMAS CARACTERÍSTICAS - UMA MATRIZ DE INTEIROS RECEBE LISTAS DE INTEIROS, UMA MATRIZ DE FLOATS RECEBE FLOATS ETC..

```
var sliceInteiros = [][]int{  
    []int{11, 20}  
    []int{15, 92}  
}
```

```
sliceFlutuantes := [][]float32{  
    []float32{20.10, 23.32}  
    []float32{2.43, 3.1415}  
}
```

MAPS





CHAVE : VALOR

PROPRIEDADES:

- Chaves não se repetem
- não permanecem necessariamente na mesma ordem de inserção / criação
- Usamos o for para percorrer

MAPS

COMANDO	FUNÇÃO
nome[chave])	retorna o valor da chave indicada
nome[chave] = valor	altera o valor da chave indicada
valor, ok := nome[chave]	verifica qual o valor da chave e se ela existe, caso não exista valor recebe 0 e ok recebe false
delete(nome, chave)	remove o elemento chave;valor

MÉTODOS

```
//declarações globais
//var nome = map[tipoChave]tipoValor
var mapVazio = map[string]int{}

var mapConteudo = map[string]int{
    "pet": 2010,
    "apresentação": 2023,
}

func main(){
    //declarações locais (iguais as globais)
    maplocVazio := map[int]bool{}

    maplocConteudo := map[int]bool{
        0: false,
        1: true,
    }
}
```

EXEMPLIFICANDO



ENTENDIDO?

PRÓXIMO NÍVEL



Utilizando a slice `pet := []int{2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019}`:

- Adicione o valor 2020
- Adicione os valores 2021, 2022 e 2023 com uma única declaração
- Demonstre a slice
- Adicione a seguinte slice: `ads := []int{2024, 2025, 2026, 2027, 2028}`
- Demonstre após o item anterior

Crie um map com chave do tipo string e valor do tipo []string. A chave deve conter nomes no formato sobrenome_nome e o valor dos hobbies favoritos da pessoa. Demonstre o map.

Utilizando a slice pet := []int{2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019}:

- Adicione o valor 2020
- Adicione os valores 2021, 2022 e 2023 com uma única declaração
- Demonstre a slice
- Adicione a seguinte slice: ads := []int{2024, 2025, 2026, 2027, 2028}
- Demonstre após o item anterior

```
package main

import "fmt"

func main() {
    // Slice
    pet := []int{2010, 2011, 2012, 2013, 2014,
2015, 2016, 2017, 2018, 2019}

    // Nova slice
    ads := append(pet[:3], pet[6:]...)

    // Demonstração
    fmt.Println(ads)
}
```

```
package main

import "fmt"

func main() {
    // Map
    mapa := map[string][]string{
        "Álvanee_Helena": []string{"DJ",
"Academia"},
        "Simoelo_Isabela": []string{"Aquarela", "Surfe"},
        "Lemáoiu_Alice": []string{"Teatro",
"Leitura"},
    }

    // Demonstração
    for chave, valor := range mapa {
        fmt.Println(chave, valor)
    }
}
```

Crie um map com chave do tipo string e valor do tipo []string. A chave deve conter nomes no formato sobrenome_nome e o valor dos hobbies favoritos da pessoa. Demonstre o map.

OBRIGADA