

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
Instituto de Ciências Exatas e Informática

Rayane Paiva Reginaldo

**A LINGUAGEM DE PROGRAMAÇÃO GO**  
**Linguagens de Programação: Trabalho Teórico Prático I**

Belo Horizonte  
2021

Rayane Paiva Reginaldo

**A LINGUAGEM DE PROGRAMAÇÃO GO**  
**Linguagens de Programação: Trabalho Teórico Prático I**

Trabalho Acadêmico apresentado ao Instituto de  
Ciências Exatas e Informática da Pontifícia  
Universidade Católica de Minas Gerais.

Professor: Prof. Marco Rodrigo Costa

Belo Horizonte  
2021

## LISTA DE FIGURAS

FIGURA 1 –	Árvore Genealógica de Go . . . . .	5
FIGURA 2 –	Exemplo de Goroutine . . . . .	9
FIGURA 3 –	Escopos em Go . . . . .	10
FIGURA 4 –	Escopos em Go - Código . . . . .	10
FIGURA 5 –	Tipos numéricos em Go . . . . .	11
FIGURA 6 –	Inferência de tipos . . . . .	11
FIGURA 7 –	Declaração, atribuição e inferência de tipo . . . . .	12
FIGURA 8 –	Exemplo de uso de ponteiros . . . . .	12
FIGURA 9 –	Exemplo de UTF-8 nativo . . . . .	13
FIGURA 10 –	Exemplo de função com múltiplos retornos . . . . .	13
FIGURA 11 –	Exemplo estrutura de repetição . . . . .	14
FIGURA 12 –	Exemplo de ponteiros em Go e C . . . . .	16
FIGURA 13 –	Exemplo de structs em Go e C . . . . .	16
FIGURA 14 –	Exemplo de uso da instrução defer . . . . .	18
FIGURA 15 –	Exemplo de uso structs . . . . .	18
FIGURA 16 –	Exemplo de uso de arrays e matrizes . . . . .	19
FIGURA 17 –	Exemplo de funcionamento dos slices . . . . .	19
FIGURA 18 –	Exemplo de retorno múltiplo de função . . . . .	20
FIGURA 19 –	Exemplo de uso das Goroutines . . . . .	21
FIGURA 20 –	Exemplo de Brute Force - Parte 1 . . . . .	22
FIGURA 21 –	Exemplo de Brute Force - Parte 2 . . . . .	23
FIGURA 22 –	Exemplo de Brute Force - Parte 3 . . . . .	24

## SUMÁRIO

<b>1 INTRODUÇÃO</b>	<b>4</b>
<b>2 HISTÓRICO E CRONOLOGIA</b>	<b>5</b>
2.1 Cronologia	6
<b>3 PARADIGMA</b>	<b>7</b>
<b>4 CARACTERÍSTICAS DA LINGUAGEM</b>	<b>8</b>
4.1 Compilada	8
4.2 Multiplataforma	9
4.3 Concorrência e goroutines	9
4.4 Suporte a closures	10
4.5 Estaticamente e fortemente tipada	11
4.6 Inferência de tipos	11
4.7 Ponteiros	12
4.8 Strings imutáveis em que o texto geralmente é codificado em UTF-8	13
4.9 Funções com múltiplos retornos	13
4.10 Todas as variáveis declaradas devem ser usadas	13
4.11 Variáveis não inicializadas tem valor padrão	14
4.12 Possui apenas um laço de repetição	14
<b>5 LINGUAGENS SIMILARES</b>	<b>16</b>
5.1 Go e C	16
5.2 Go e outras linguagens	17
<b>6 EXEMPLOS DE PROGRAMAS</b>	<b>18</b>
6.1 Uso da instrução defer	18
6.2 Uso de structs	18
6.3 Uso de arrays e matrizes	19
6.4 funcionamento dos slices	19
6.5 Uso de funções com retorno múltiplo	20
<b>7 ESTUDO DE CASO</b>	<b>21</b>
7.1 Goroutines	21
7.2 Brute-Force	22
<b>8 CONSIDERAÇÕES</b>	<b>25</b>
<b>REFERÊNCIAS</b>	<b>27</b>

## 1 INTRODUÇÃO

O projeto inicial da linguagem de programação **Go** foi criado em setembro de 2007 por três funcionários do Google, Robert Griesemer (Cientista da Computação), Rob Pike (Engenheiro de Software) e Ken Thompson (Cientista da Computação), oficialmente a linguagem foi lançada em novembro de 2009. Segundo o *site oficial*,<sup>(1)</sup>, Go é uma linguagem de programação de código aberto que facilita a criação de softwares simples, confiáveis e eficientes.

É uma linguagem muito semelhante a linguagem C, e muitas vezes pintada como uma "linguagem do tipo C" ou mencionada como "C para o século XXI", porém a linguagem busca simplicidade e performance diferente de C e C++ quem possuem performance, porém podem gerar certa complexidade na hora do desenvolvimento. Go propõe conciliar boas ideias de muitas outras linguagens e como busca produtividade também evita várias outras ideias que poderiam tornar o desenvolvimento de código mais complexo e não confiável.

É uma linguagem baseada em trabalhos feitos no sistema operacional Inferno (1996), sistema usado para criação e suporte a serviços distribuídos. Sendo a concorrência uma consequência do uso de sistemas distribuídos, e Go tendo surgido pela motivação de resolver problemas nesse tipo de sistema a linguagem é apresentada como sendo do paradigma concorrente, mas hoje é dita como uma linguagem de multipropósito e é muito usada em coisas para web.

Por ser uma linguagem moderna, ela foi lançada em 2009, ou seja, tem apenas 12 anos, na implementação da linguagem foi possível agrupar muitas das evoluções computacionais que não existiam ainda no tempo que as outras linguagens mais antigas estavam sendo criadas.

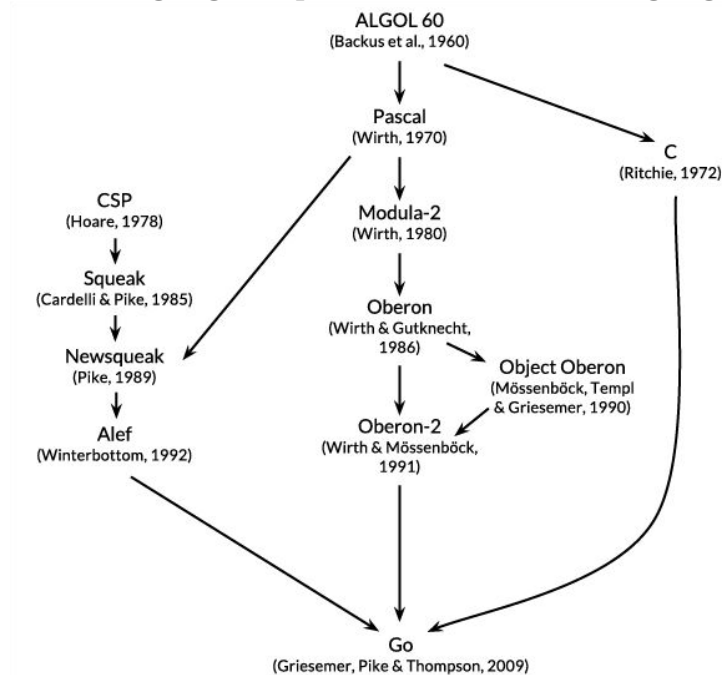
## 2 HISTÓRICO E CRONOLOGIA

Como dito anteriormente, (??) Go muitas vezes é descrita como uma linguagem semelhante a linguagem C. Podemos destacar como influencias de C os tipos básicos de dados, a passagem de parâmetro por valor, ponteiros e especialmente os programas compilados. Porém existem outras linguagens ascendentes que também influenciaram na criação e no desenvolvimento da linguagem. Como destacado em (2): "Modula-2 inspirou o conceito de pacote, Oberon eliminou a distinção entre arquivos de interface de módulos e arquivos de implementação de módulos. Oberon 2 influenciou a sintaxe de pacotes, importações e declarações, e Object Oberon forneceu a sintaxe para declarações de métodos."

A linguagem nasceu pela necessidade de resolver problemas com alguns sistemas de software no Google. A empresa no momento se deparava com um grande crescimento da complexidade no desenvolvimento e manutenção do código. Rob Pike, um dos desenvolvedores da linguagem, cita que "a complexidade é multiplicativa", ou seja, ao reparar um problema dos sistemas com um código mais complexo aos poucos deixa outras partes do sistema mais complexas. Era necessária uma linguagem que simples para garantir estabilidade, segurança e coerência ao sistema a medida que o mesmo estivesse sendo desenvolvido.

Linguagens que tiveram influencia na criação da linguagem:

**Figura 1 – Linguagens que influenciaram a linguagem Go**



Fonte: DONOVAN E KERNIGHAN, 2017

## 2.1 Cronologia

Um pouco da Cronologia da linguagem:

- Setembro de 2007 - Projeto inicial por Robert Grieseme, Rob Pike e Ken Thompson;
- Janeiro de 2008 - Primeira versão gerando código em C (compilador gc);
- Novembro de 2009 - Divulgada para o público;
- 2010 - Começou a ser adotado por outros programadores;
- Março 2012 - Go 1 correções de bugs e inconsistências e melhorias na portabilidade;
- Agosto 2015 - Go 1.5 compilador escrito inteiramente em Go;
- Fevereiro 2021 - Go 1.16 mudanças na implementação da cadeia de ferramentas, tempo de execução e bibliotecas.

Um pouco do Histórico de versão:

- Go 1 - Março 2012;

### 3 PARADIGMA

Tem se tornado comum, em linguagens contemporâneas, o uso de múltiplos paradigmas em uma mesma linguagem. A linguagem Go tem recursos e atributos de linguagens como C (imperativa), Java(Orientada a Objetos) e LISP(funcional). Apesar disso o paradigma **concorrente** e imperativo são os que predominam em Go.

O *paradigma concorrente* (3) está presente no desenvolvimento de programas que fazem uso de execução simultânea de varias tarefas, que podem ser executadas por um único processador ou múltiplos processadores em um único equipamento ou em processadores distribuídos em uma rede.

Sendo multi-paradigma, a linguagem torna possível explorar variados recursos de cada um dos paradigmas que ela incorpora. Porem, mesmo apresentando aspectos da orientação a objetos, não podemos afirmar que Go é de fato orientada a objetos.(4) A linguagem permite sim um estilo de programação orientado a objetos, mas não existe hierarquia de tipos e a ideia de interface em Go é usada de uma maneira mais geral, não existe a possibilidade de ter classes derivadas de uma super classe como em Java, ou seja não há herança em Go.

A concorrência faz com que a linguagem uma ótima opção para criação de programas que rodam em sistemas distribuídos e ferramentas para programadores, mas ela também pode ser usada em aplicações moveis e aprendizagem de maquina por exemplo, é também uma ótima opção de linguagem de scripting pois por ser fortemente tipada, diferente da maioria das linguagens usadas para esse fim, ela oferece mais segurança. Além disso, Go geralmente executa seus programas mais rápido do que alguns programas escritos em linguagens dinâmicas.

No paralelismo duas ou mais tarefas são executadas ao mesmo tempo em núcleos diferentes. Na concorrência, duas ou mais tarefas são **enviadas** para execução, mas a execução destas tarefas podem ser feitas em um ou mais núcleos. Além disso as tarefas vão concorrer com umas com as outras por uma fatia do tempo de execução neste núcleo.



## 4 CARACTERÍSTICAS DA LINGUAGEM

As principais características da linguagem Go são:

- Compilada;
- Multiplataforma;
- Concorrência;
- Suporte a closures
- Estaticamente e fortemente tipada;
- Inferência de tipos;
- Ponteiros
- Coletor de lixo;
- Dependências explícitas;
- Goroutines;
- Funções com múltiplos retornos;
- Desvios incondicionais;
- Não usa ";"
- Legibilidade;
- Sintaxe simples;
- Confiabilidade;
- Excelente pacote padrão;
- Tem seu próprio Go Runtime;
- Strings imutáveis em que o texto geralmente é codificado em utf8;
- Todas as variáveis declaradas devem ser usadas;
- Variáveis não inicializadas tem valor padrão;
- Possui apenas um laço de repetição;
- E varias outras.

Nas seções a seguir algumas destas características serão descritas e/ou exemplificadas.

### 4.1 Compilada

Para executar um programa em Go, devemos compilar o seu código fonte para gerar um executável. Go é Compilada e conversa diretamente com a arquitetura do computador onde o programa escrito esta sendo executado.

## 4.2 Multiplataforma

Linguagens multiplataforma podem ser executadas em diferentes ambientes. Além, de ser multiplataforma Go também garante a retrocompatibilidade, ou seja, todo código escrito em versões antigas de Go vão funcionar sem problemas nas versões mais novas da linguagem.

## 4.3 Concorrência e goroutines

A concorrência é indicada pela palavra chave `go` precedendo a execução de uma função. Quando executamos uma função precedido da palavra chave `go`, dizemos que temos uma goroutine. (5)

**Figura 2 – Goroutine**

A code editor window with a dark background and light-colored text. The code is written in Go and demonstrates goroutines. It includes a package declaration, imports for 'fmt' and 'time', a function 'say' that prints a string with a delay, and a 'main' function that calls 'say' twice, once with 'go' to run it concurrently.

```
package main

import (
    "fmt"
    "time"
)

func say(s string){
    for i:=0 ; i<=5 ; i++){
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

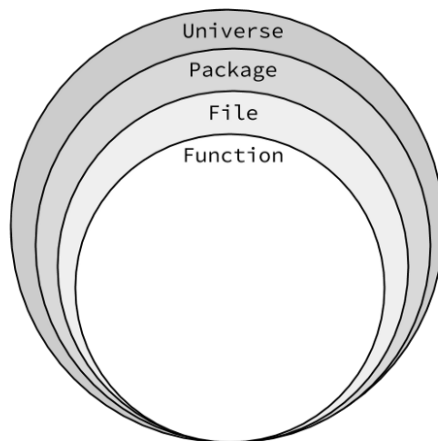
func main(){
    go say("world")
    say("hello")
}
```

Fonte: CARVALHO, 2015

#### 4.4 Suporte a closures

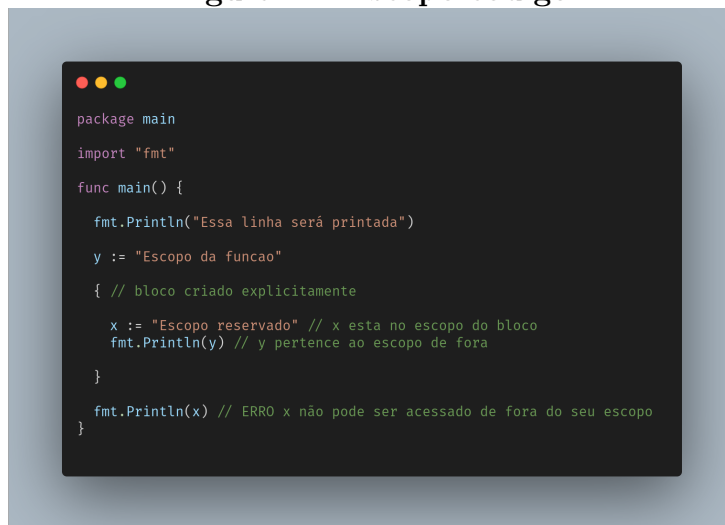
Closure é quando uma função é capaz de "lembrar" e acessar seu escopo léxico mesmo quando ela está sendo executada fora de seu escopo léxico.

**Figura 3 – Escopo da linguagem**



Fonte: MAGNUM, 2016

**Figura 4 – Escopo código**



Fonte: MAGNUM, 2016

Podemos ver na figura 4 que dentro de um bloco temos acesso a variáveis de um escopo exterior, porém de fora de um escopo de bloco não podemos acessar variáveis que ele contém.

## 4.5 Estaticamente e fortemente tipada

Em Go possui um sistema de tipos que é verificado no momento da compilação e não seja possível para o programador contornar as restrições impostas pelo sistema de tipos.

Os tipos booleanos em Go tem os valores **true** e **false**, as strings em Go são possivelmente uma sequência de bytes e são imutáveis, os tipos numéricos estão expostos na figura 5.

**Figura 5 – tipos Numéricos**

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

Fonte: Go, 2012

## 4.6 Inferência de tipos

Go trás características de linguagens dinamicamente tipadas, como Python por exemplo, mas mantém a eficiência e a segurança de uma linguagem como C ou C++. Nas figuras 6a e 6b ambas as formas de declaração de variáveis vão resultar no mesmo tipo. E ainda existe uma terceira forma de declarar e atribuir as variáveis como pode ser visto na figura 7.

No caso das variáveis do tipo float, que em Go podem ser de dois tipos, **float32** para números de até 32 bits e **float64** para números de até 64 bits, caso não seja especificado o tipo, por padrão, Go define a variável como sendo do tipo **float64**.

A declaração curta de variável usando o operador **:=** só pode ser feita dentro do escopo de funções.

**Figura 6 – Exemplo de inferência de tipos**

(a) Sem inferência.

(b) Com inferência.

```
package main

func main(){
    var nome string = "Bruce Wanyne"
    var idade int = 32
    var altura float32 = 1.88
}
```

```
package main

func main(){
    var nome = "Bruce Wanyne"
    var idade = 32
    var altura = 1.88
}
```

Fonte: SOUZA, 2018

**Figura 7 – Declarando, atribuindo e inferindo tipo**A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays Go code for a package named 'main' with a 'main' function. Inside the function, three variables are declared and assigned: 'nome' to 'Bruce Wanyne', 'idade' to '32', and 'altura' to '1.88'.


```
package main

func main(){
    nome := "Bruce Wanyne"
    idade := 32
    altura := 1.88
}
```

**Fonte: PAIVA, 2021**

## 4.7 Ponteiros

Quando atribuímos valor a uma variável, esse valor é armazenado em um endereço de memória no computador. Para visualizar o endereço de memória de uma variável podemos usar o operador `&` em Go. E para definir um ponteiro em Go usamos o operador `*` seguido do tipo do valor que será armazenado no endereço para o qual o ponteiro irá apontar. Para acessar o valor armazenado no endereço do ponteiro usamos o também o operador `*` seguido do nome do ponteiro.(7)

**Figura 8 – Uso de ponteiros**A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays Go code for a package named 'main' that imports the 'fmt' package. The 'main' function declares a string variable 'nome' with the value 'Rayane Paiva'. It then declares a pointer variable 'ponteiro' of type '\*string' and assigns it the address of 'nome' using the '&' operator. It prints the memory address of 'nome', the address stored in 'ponteiro', and the content of 'ponteiro' (which points to 'nome'). Then, it changes the value of the pointer 'ponteiro' to 'Leticia' and prints the content of 'ponteiro' again, which now points to the new memory location.

```
package main

import "fmt"

func main() {

    nome := "Rayane Paiva"

    var ponteiro *string
    ponteiro = &nome

    fmt.Println("Endereço de memória: ", &nome)
    fmt.Println("Endereço no ponteiro: ", ponteiro)
    fmt.Println("Conteúdo do ponteiro: ", *ponteiro)

    *ponteiro = "Leticia"


    fmt.Println("Conteúdo do ponteiro: ", *ponteiro)
}
```

**Fonte: PAIVA, 2021**

#### 4.8 Strings imutáveis em que o texto geralmente é codificado em UTF-8

Go trata unicode de modo nativo, portanto é capaz de processar textos em todas as línguas do mundo. No exemplo de código a seguir a saída na tela será exatamente a que esta escrita na chamada da função **fmt.Println**.

**Figura 9 – Strings em Go**



```
package main

import (
    "fmt"
)

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

Fonte: (2)

#### 4.9 Funções com múltiplos retornos

As função em Go podem ter múltiplos retornos. Ao retornar os valores da função, é necessário informar todos os retornos. Os retornos e seus tipos são informados após os parênteses dos parâmetros e antes da chave de abertura das instruções.

**Figura 10 – Uso de múltiplos retornos em uma função**



```
package main

import "fmt"

func troca(a, b string) (string, string){
    return b, a
}

func main() {

    nome1 := "Rayane"
    nome2 := "Leticia"

    nome1, nome2 =: troca(nome1, nome2)

    fmt.Println("Conteudo de nome1: ", nome1)
    fmt.Println("Conteudo de nome2: ", nome2)
}
```

Fonte: PAIVA, 2021

#### 4.10 Todas as variáveis declaradas devem ser usadas

Não é permitido pela linguagem fazer a declaração de variáveis que não serão usadas no código. Esta é uma convenção da linguagem e caso não seja seguida o código gera erro

ao tentar compilar.


#### 4.11 Variáveis não inicializadas tem valor padrão

Caso uma variável seja apenas declarada e não receba valor durante a declaração e em nenhum outro momento, quando ela for usada em algum momento no código o seu valor armazenado será 0 para inteiros, 0.0 para os tipos de float e para strings.

#### 4.12 Possui apenas um laço de repetição

GO possui apenas um laço de repetição, o **for**, e ele é usado para as mesmas funcionalidades que os laços for e while em Java por exemplo.

Figura 11 – Uso do laço for



```
package main

import "fmt"

func main() {

    // for comum
    for i := 0 ; i < 3 ; i++ {
        fmt.Println("%d",i)
    }

    // while
    j := 0
    for j < 3 {
        j += 1
    }
    fmt.Println("%d",j)

    // loop infinito
    for {
        fmt.Println("%d",j)
    }
}
```

Fonte: PAIVA, 2021

Características de outras linguagens que não podem ser encontradas em Go:

- Herança;
- Classes;
- Exceptions;
- Sobrecarga de métodos;
- Conversões numéricas implícitas;
- Construtores ou destrutores;
- Valores default para parâmetros;
- Tipos genéricos;



## 5 LINGUAGENS SIMILARES

### 5.1 Go e C

Como já foi dito, Go é uma linguagem conhecida por ser do tipo C like. É facilmente perceptível a similaridade de Go com C em sua sintaxe:

**Figura 12 – Ponteiros em Go e C**

(a) Golang.

(b) C.

```
import "fmt"

func main() {

    num := 3
    *ponteiro := &num
    fmt.Printf("Numero: %d", num)
    fmt.Printf("Endereco de memoria: %v", ponteiro)
}
```

```
#include<stdio.h>

int main() {

    int num = 3;
    int *ponteiro = &num;
    printf("Numero: %d", num);
    printf("Endereco de memoria: %v", ponteiro);
}
```

Fonte: PAIVA, 2021

**Figura 13 – Structs em Go e C**

(a) Golang.

(b) C.

```
type Pessoa struct {

    CPF string
    nome string
    idade int
}
```

```
typedef struct Pessoa {

    char CPF[11];
    char nome[80];
    int idade;
}
```

Fonte: PAIVA, 2021

Além da sintaxe, C e Go também podem ser compiladas em diferentes arquiteturas de hardware ou de software. É possível usar tanto Go como C em Mac ou PC, em Linux ou em Windows. Ambas as linguagens são também de tipagem estática, ou seja, possuem uma verificação de tipos feita no código fonte pelo processo de compilação. Go e C tem também funções, e as duas linguagens são de uso geral. Existem varias outras similaridades entre Go e C, inclusive, um dos criadores de Go, o Cientista da computação Ken Thompson, criou a linguagem B que foi antecessora a linguagem C.

De acordo com DONOVAN E KERNIGHAN, 2017, de c, Go herdou a sintaxe de suas expressões, as instruções de controle de fluxo, tipos básicos de dados, passagem de parâmetros por valor, ponteiros e, acima de tudo, a ênfase de C em programas compilados que geram código de maquina eficiente e cooperem naturalmente com as abstrações dos sistemas operacionais atuais.

C e Go tem as seguintes características em comum:

- Portabilidade;
- Modularidade;
- Sistema de tipos;
- Imperativa;
- Uso geral;
- E algumas outras.

## 5.2 Go e outras linguagens

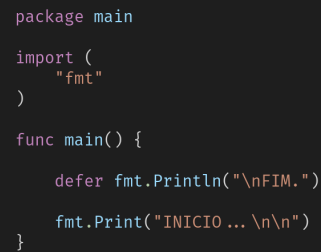
Java, C++ e Python, por exemplo, são outros exemplos de linguagens que são portáteis assim como Go. Vemos outras pequenas semelhanças em ALGOL e Pascal também como por exemplo o uso do operador `:=` para fazer associações. O escopo léxico de Go e as funções aninhadas são semelhantes ao que vemos em Scheme.

Por ser uma linguagem nova, Go, tem muito de outras linguagens dentro de si o que faz com que ela tenha pontos de similaridade com diversas linguagens que já existem a mais tempo.

## 6 EXEMPLOS DE PROGRAMAS

### 6.1 Uso da instrução defer

Figura 14 – Instrução defer



```
package main

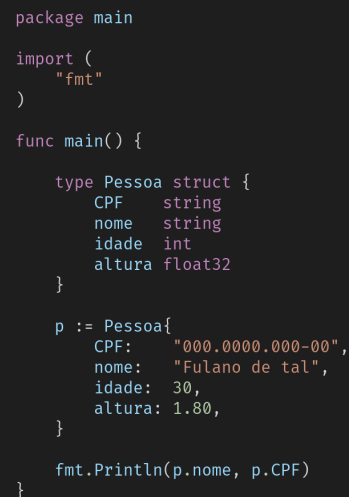
import (
    "fmt"
)

func main() {
    defer fmt.Println("\nFIM.")
    fmt.Print("INICIO ... \n\n")
}
```

Fonte: PAIVA, 2021

### 6.2 Uso de structs

Figura 15 – Struct



```
package main

import (
    "fmt"
)

func main() {
    type Pessoa struct {
        CPF    string
        nome   string
        idade  int
        altura float32
    }

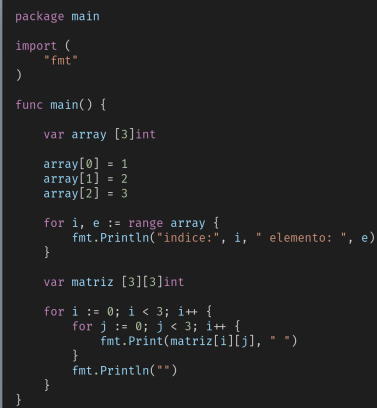
    p := Pessoa{
        CPF:    "000.0000.000-00",
        nome:   "Fulano de tal",
        idade:  30,
        altura: 1.80,
    }

    fmt.Println(p.nome, p.CPF)
}
```

Fonte: PAIVA, 2021

### 6.3 Uso de arrays e matrizes

Figura 16 – Array e Matriz

A screenshot of a Go code editor window with a dark background and light-colored text. The code defines a package 'main', imports the 'fmt' package, and implements a 'main' function. Inside 'main', it declares an integer array 'array' of size 3, initializes its elements to 1, 2, and 3, and then iterates over them to print each element with its index. It also declares a 3x3 integer matrix 'matriz', iterates over each element, and prints the matrix contents in a formatted grid.

```
package main

import (
    "fmt"
)

func main() {

    var array [3]int

    array[0] = 1
    array[1] = 2
    array[2] = 3

    for i, e := range array {
        fmt.Println("indice:", i, " elemento: ", e)
    }

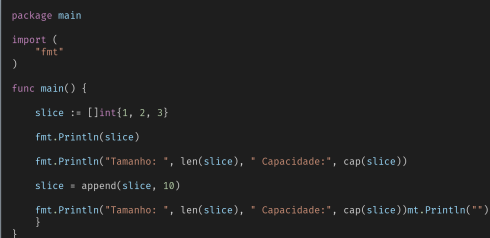
    var matriz [3][3]int

    for i := 0; i < 3; i++ {
        for j := 0; j < 3; j++ {
            fmt.Print(matriz[i][j], " ")
        }
        fmt.Println("")
    }
}
```

Fonte: PAIVA, 2021

### 6.4 funcionamento dos slices

Figura 17 – Slice

A screenshot of a Go code editor window with a dark background and light-colored text. The code defines a package 'main', imports the 'fmt' package, and implements a 'main' function. Inside 'main', it declares a slice 'slice' containing the integers 1, 2, and 3, prints the slice, and then appends the value 10 to it. Finally, it prints the slice again to show the updated state.

```
package main

import (
    "fmt"
)

func main() {

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

    fmt.Println(slice)

    fmt.Println("Tamanho: ", len(slice), " Capacidade:", cap(slice))


    slice = append(slice, 10)

    fmt.Println("Tamanho: ", len(slice), " Capacidade:", cap(slice))
    mt.Println("")
}
```

Fonte: PAIVA, 2021

## 6.5 Uso de funções com retorno múltiplo

**Figura 18 – Função com retorno múltiplo**



```
package main

import (
    "fmt"
)

func main() {
    nome, idade := exRetornoMultiplo()
    fmt.Println("Nome: ", nome, " idade: ", idade)
}

func exRetornoMultiplo() (string, int) {
    return "Rayane", 27
}
```


Fonte: PAIVA, 2021

## 7 ESTUDO DE CASO

### 7.1 Goroutines

**Goroutine** é uma linha de execução gerenciada pelo sistema de execução da própria linguagem Go. A expressão: *go foo(a, b, c)* inicia a execução de uma nova goroutine. A avaliação da função *foo* acontece na rotina corrente e sua execução acontece em uma nova goroutine.

Figura 19 – Goroutine



```
package main

import (
    "fmt"
    "strconv"
    "sync"
    "time"
)

var wg sync.WaitGroup

func main() {
    executaTarefa()
}

func tarefa(s string) {
    defer wg.Done()

    for i := 0; i < 3; i++ {
        time.Sleep(1 * time.Second)
        fmt.Println("Tarefa ", s)
    }

    fmt.Println("Tarefa CONCLUIDA")
}

func executaTarefa() {
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go tarefa(strconv.Itoa(i))
    }

    wg.Wait()

    fmt.Println("execução CONCLUIDA")
}
```

Fonte: CODINHOTO, 2017

## 7.2 Brute-Force

O brute-force (força-bruta), é um método de tentativa e erro para quebrar senhas, ou seja, usamos diferentes senhas até que a senha correta seja encontrada. Esse é, até hoje, a maneira mais usada para quebrar senhas.

**Figura 20 – Brute-Force - Parte 1**



Fonte: APOLINARIO, 2021

Figura 21 – Brute-Force - Parte 2

```
func abrirArquivoZip() {  
    arquivos, err := zip.OpenReader(zipPath)  
    if err != nil {  
        panic(err.Error())  
    }  
    defer arquivos.Close()  
    arquivo := arquivos.File[0]  
    if arquivo.IsEncrypted() {  
        listaSenhas := obterListaDeSenhas(passPath)  
        start := time.Now()  
        for i := 0; i < len(listaSenhas); i++ {  
            senhaAtual := listaSenhas[i]  
            arquivo.SetPassword(senhaAtual)  
            r, err := arquivo.Open()  
            if err != nil {  
                fmt.Println(err.Error())  
                break  
            }  
            buf, err := io.ReadAll(r)  
            // se não tiver erro na leitura.  
            if err != nil {  
                continue  
            }  
            defer r.Close()  
            if buf != nil {  
                fmt.Printf("\nSenha encontrada: %v\n", senhaAtual)  
                fmt.Printf("localizada na linha: %v\n", i+1)  
                break  
            }  
        }  
        fmt.Printf("\nTempo brute-force: %v\n", time.Since(start))  
    }  
}
```

Fonte: APOLINARIO, 2021



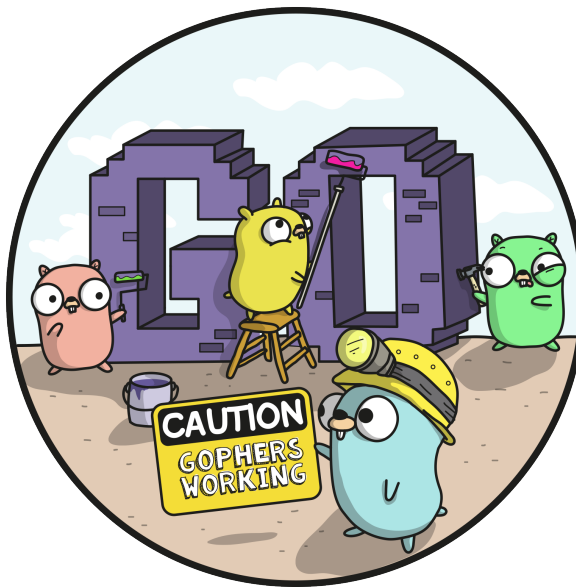
Figura 22 – Brute-Force - Parte 3

```
func obterListaDeSenhas(caminhoArquivo string) (senhas []string) {  
    arquivo, err := os.Open(caminhoArquivo)  
  
    if err != nil {  
        panic(err.Error())  
    }  
  
    scanner := bufio.NewScanner(arquivo)  
    scanner.Split(bufio.ScanLines)  
  
    start := time.Now()  
    for scanner.Scan() {  
        senhas = append(senhas, scanner.Text())  
    }  
    fmt.Printf("\nTempo de leitura arquivo: %v\n", time.Since(start))  
  
    defer arquivo.Close()  
  
    arquivoStatus, err := os.Stat(caminhoArquivo)  
  
    if err != nil {  
        panic(err.Error())  
    }  
  
    fmt.Printf("\nNome Arquivo: %v\n", arquivoStatus.Name())  
    fmt.Printf("Tamanho Arquivo: %v\n", arquivoStatus.Size()/1024)  
    fmt.Printf("Quantidade de senhas: %v\n", len(senhas))  
  
    return  
}
```

Fonte: APOLINARIO, 2021

## 8 CONSIDERAÇÕES

Go é uma linguagem muito eficiente quando falamos de concorrência, mas não deixa a desejar em outras áreas, como já foi mencionado a linguagem é de uso geral. Além disso é uma linguagem que tem convenções já muito bem definidas, o que faz com que discussões desnecessárias sobre as melhores formas de se programar sejam evitadas. Não é uma linguagem muito legível, apesar de que na sua criação um dos propósitos era a redução do código necessário para se escrever um algoritmo eficaz, acabamos por ter que escrever muito durante o tratamento de erros, apesar disso, ainda é uma linguagem de fácil aprendizado e muito poderosa.





## REFERÊNCIAS

- 1 GO, E. de D. *Go web site*. 2012. Site. Disponível em: <<https://golang.org/>>. Acesso em: (acesso: 03.04.2021). 4, 11
- 2 DONOVAN, A.; KERNIGHAN, B. *A linguagem de programação Go*. NOVATEC, 2017. Livro. ISBN 9788575225462. Disponível em: <<https://books.google.com.br/books?id=2kZGDgAAQBAJ>>. 5, 13, 16
- 3 WIKIPEDIA. Site, *Programação Concorrente*. 2020. Disponível em: <<https://bit.ly/3p0PxFO>>. Acesso em: (acesso: 07.05.2021). 7
- 4 LIMA, J. O. *Go é orientado a objetos ?* 2020. Artigo de Blog. Disponível em: <<https://dev.to/jeffotoni/go-e-orientada-a-objetos-3gf9>>. Acesso em: (acesso: 10.05.2021). 7
- 5 PAIVA, R. 2021. Figura elaborada pelo autor. 9, 12, 13, 14, 16, 18, 19, 20
- 6 CARVALHO, S. G. *Go Lang - A linguagem do Google*. 2015. PDF. Disponível em: <<https://www.ime.usp.br/~amaris/mac-5742/reports/GoLang.pdf>>. Acesso em: (acesso: 07.05.2021). 9
- 7 MAGNUM, L. *Iniciando em Go — Ponteiros*. 2016. Artigo de Blog. Disponível em: <<https://lucasmagnum.medium.com/iniciando-em-go-ponteiros-2d990318c0fb>>. Acesso em: (acesso: 15.04.2021). 10, 12
- 8 SOUZA, E. F. *GoLang — Simplificando a complexidade*. 2018. Artigo de Blog. Disponível em: <<https://medium.com/trainingcenter/golang-d94e16d4b383>>. Acesso em: (acesso: 17.04.2021). 11
- 9 GOLANG. *Companies currently using Go throughout the world*. 2020. Publicação no github. Disponível em: <<https://github.com/golang/go/wiki/GoUsers>>. Acesso em: (acesso: 30.04.2021).
- 10 NIEVA, H.; MELO, R. *Votações na Globo: Processando 1 bilhão de votos*. 2020. Video. Disponível em: <<https://youtu.be/rnUOmOzwm9o>>. Acesso em: (acesso: 07.05.2021).
- 11 LIMA, J. O. *Go: A linguagem que mais cresce para backend*. 2020. Artigo de Blog. Disponível em: <<https://jeffotoni.medium.com/porque-amamos-go-ea27ec919c53>>. Acesso em: (acesso: 10.05.2021).
- 12 PIKE, R. *Go at Google: Language Design in the Service of Software Engineering*. 2012. Keynote Talk. Disponível em: <[https://talks.golang.org/2012/splash.article#TOC\\_3](https://talks.golang.org/2012/splash.article#TOC_3)>. Acesso em: (acesso: 10.04.2021).
- 13 APOLINARIO, I. Palestra, *Linguagem Go (Golang)*. 2021. Video. Disponível em: <<https://youtu.be/37uG6C7RmkI>>. Acesso em: (acesso: 03.04.2021). 22, 23, 24
- 14 OLIVEIRA, M. *Conheça o Inferno*. 2020. Artigo de Blog. Disponível em: <<https://terminalroot.com.br/2020/09/conheca-o-inferno.html>>. Acesso em: (acesso: 30.04.2021).

15 CODINHOTO, E. *Golang - Aula 14 (Goroutines)*. 2017. Video. Disponível em: <<https://youtu.be/GPz49npOKzQ>>. Acesso em: (acesso: 26.04.2021). 21