GO

- Fernando
- Guilherme
- Igor
- João Felipe Gobeti

Introdução

Introdução

- Criada pela Google em 2009.
- Surgiu para resolver complexidades de sistemas distribuídos, processadores multinúcleos, etc.
- Foco em programação concorrente.

Introdução - Características

- Linguagem Compilada.
- Open source
- Fortemente e estaticamente tipada.
- Coleta de Lixo.
- Suporte a closures.
- Dependência explícita
- Funções com múltiplos retornos
- Ponteiros

Introdução - O que não tem

- Tratadores de exceção.
- Classes
- Herança
- Generics
- Assertions
- Sobrecarga de métodos

Introdução - Exemplo e Compilação

```
package main
import "fmt"
func main() {
fmt.Printf("Olá, mundo!\n");
fmt.Printf("Esse é o
seminário de Go!");
}
```

```
joaogcalen@LinuxDoRoberval ~/go/src/hello$ go run hello
Olá, Mundo!
Esse é o seminário de Go!
```

Amarrações

Amarrações - Palavras Chaves

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

Amarrações - Escopo

- Blocos léxicos
- Declaração do bloco determina o escopo
- Um nome pode conter múltiplas declarações, desde que em blocos distintos
- Compilador procura a referência mais interna a um dado nome

Amarrações - Escopo - Exemplo

```
func main() {
    x := "hello!"
    for i := 0; i < len(x); i++ {
        x := x[i]
        if x != '!' {
            x := x + 'A' - 'a'
            fmt.Printf("%c", x) // "HELLO"
        }
    }
}
```

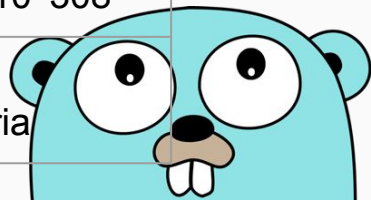
Amarrações - Escopo - Exemplo 2

```
if x := f(); x == 0 {
    fmt.Println(x)
} else if y := g(x); x == y {
    fmt.Println(x, y)
} else {
    fmt.Println(x, y)
}
fmt.Println(x, y) // Erro de compilação
```

Valores e Tipos de Dados

Tipos Primitivos - Números

Tipo	Descrição	Intervalo
uint8, uint16, uint32, uint64	Inteiros positivos (unsigned int)	0 a (2^8 -1), 0 a (2^16 -1),0 a (2^32 -1),0 a (2^64 -1)
int8, int16, int32, int64	Números inteiros (signed)	-2^7 a (2^7 -1), -2^15 a (2^15 -1), -2^31 a (2^31 -1), -2^63 a (2^63 -1)
float32, float64	Números reais	1.5*10^-45 a 3.4*10^38, 5.0*10^-324 a 1.7*10^308
complex64, complex128	Números complexos	float32 e float64 com parte imaginária



Tipos Primitivos - Outros

Tipo	Descrição
string	Cadeia de caracteres
bool	Valores booleanos (true ou false)

Tipos primitivos - Dependentes da arquitetura

- int
- float
- uint
- uintptr

Tipos primitivos - Alias

- byte
- rune

Tipos primitivos - Exemplo

```
package main
import "fmt"
var (
    maxInt8    int8    = 1<<7 - 1
    maxInt16   int16   = 1<<15 - 1
    maxInt32   int32   = 1<<31 - 1
    maxInt64   int64   = 1<<63 - 1
    maxUint8   uint8   = 1<<8 - 1
    maxUint16  uint16  = 1<<16 - 1
    maxUint32  uint32  = 1<<32 - 1
    maxUint64  uint64  = 1<<64 - 1
    maxInt     int     = 1<<63 - 1
    maxUint    uint    = 1<<64 - 1
    maxUintptr uintptr = 1<<64 - 1
    aliasbyte  byte    = maxUint8
    aliasrune  rune    = maxInt32
)
```

```
func main() {
    fmt.Printf("Max int8:  %d\n", maxInt8)
    fmt.Printf("Max int16:  %d\n", maxInt16)
    fmt.Printf("Max int32:  %d\n", maxInt32)
    fmt.Printf("Max int64:  %d\n", maxInt64)
    fmt.Printf("Max uint8:  %d\n", maxUint8)
    fmt.Printf("Max uint16:  %d\n", maxUint16)
    fmt.Printf("Max uint32:  %d\n", maxUint32)
    fmt.Printf("Max uint64:  %d\n", maxUint64)
    fmt.Printf("Max int:  %d\n", maxInt)
    fmt.Printf("Max uint:  %d\n", maxUint)
    fmt.Printf("Max uintptr:  %d\n", maxUintptr)
    fmt.Printf("Max byte:  %d\n", aliasbyte)
    fmt.Printf("Max rune:  %d\n", aliasrune)
}
```

Tipos primitivos - Exemplo

```
joaogcalen@LinuxDoRoberval ~/go/src/tipos_primitivos_numericos$ go run tipos_primitivos_numericos.go
Max int8:  127
Max int16:  32767
Max int32:  2147483647
Max int64:  9223372036854775807
Max uint8:  255
Max uint16:  65535
Max uint32:  4294967295
Max uint64:  18446744073709551615
Max int:  9223372036854775807
Max uint:  18446744073709551615
Max uintptr:  18446744073709551615
Max byte:  255
Max rune:  2147483647
```

Tipos primitivos - Exemplo

```go
package main

import (
    "fmt"
)

var (
    testeFloat32    float32    = 1.5e-45
    teste2Float32   float32    = 1.5e-46
    testeFloat64    float64    = 4e-324
    teste2Float64   float64    = 5e-325
    testeComplexo   complex64  = complex(testeFloat32, 0) - 3.4e+38i
    teste2Complexo  complex64  = complex(teste2Float32, 0) - 3.4e+38i
)
```

```go
func main() {
    fmt.Print("Teste1 float32: ")
    fmt.Println(testeFloat32)
    fmt.Print("Teste2 float32: ")
    fmt.Println(teste2Float32)
    fmt.Print("Teste1 float64: ")
    fmt.Println(testeFloat64)
    fmt.Print("Teste2 float64: ")
    fmt.Println(teste2Float64)
    fmt.Print("Teste1 complex64: ")
    fmt.Println(testeComplexo)
    fmt.Print("Teste2 complex64: ")
    fmt.Println(teste2Complexo)
}
```

Tipos primitivos - Exemplo 2

```
joaogcalen@LinuxDoRoberval ~/go/src/tipos_primitivos_numericos2$ go run tipos_primitivos_numericos2.go
Teste1 float32: 1e-45
Teste2 float32: 0
Teste1 float64: 5e-324
Teste2 float64: 0
Teste1 complex64: (1e-45-3.4e+38i)
Teste2 complex64: (0-3.4e+38i)
```

Tipos Compostos

- Arrays
- Slices
- Maps
- Ponteiros
- Structs e interfaces

Tipos Compostos - Arrays

- Arranjo com número pré-definido de elementos
- Um array [n]T é um array com n elementos do tipo T
- Tamanho não pode ser redefinido

```
package main
func main() {
   var array1 [2]string
   array1[0] = "Hello"
   array1[1] = "World!"

   array2 := [2]string{"Hello", "World!"}

   array3 := [...]string{"Hello", "World!"}

   var matriz [2][2]int
}
```

Tipos Compostos - Slices

- Segmento de um Array
- Tamanho variável
- Todo slice está associado a um array
- Pode ser criado pela função make(tipo, tamanho, capacidade)
- Função append para inserir novos elementos no slice

Tipos Compostos - Slices

Abaixo, os slices 4, 5, 6 e 7 possuem os mesmos valores

```
var slice1 []float64
slice2 := make([]float64, 5)
slice3 := make([]float64, 5, 10)
slice4 := []int{1, 2, 3, 4, 5}
slice5 := slice4[0:5]
slice6 := slice4[0:]
slice7 := slice4[:5]
```

Tipos Compostos - Slices - Exemplo

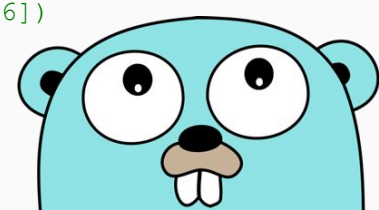
```go
package main

import "fmt"

func main() {
    var slice1 []float64
    slice2 := make([]float64, 5)
    slice3 := make([]float64, 5, 10)
    slice4 := []int{1, 2, 3, 4, 5}
    slice2[3] = 1
    slice3[3] = 1
    //slice2[5] = 1
    //slice1[0] = 1
    fmt.Println(slice1)
    fmt.Println(slice2)
    fmt.Println(slice3)
    fmt.Println(slice4)
}
```

```
joaogcalen@LinuxDoRoberval ~/go/src/slices$ go run
slices.go
[]
[0 0 0 1 0]
[0 0 0 1 0]
[1 2 3 4 5]
```

Tipos Compostos - Slices - Exemplo 2

```
package main
import "fmt"
var (
   todosOsSeminarios = [10]string{"Python",
"C#", TypeScript", "Scala", "Haskell", "Go",
"Kotlin", "Lua", "Dart", "Visual Basic"}
   seminarios        = []string{"Python", "C#",
"TypeScript", "Scala", "Haskell"}
   apresentados      = todosOsSeminarios[:5]
   apresentando      = todosOsSeminarios[5:6]
   apresentarao      = todosOsSeminarios[6:]
   todasAsFatias     []string
)
```

```
func main() {
   seminarios = append(seminarios, "Go")
   todasAsFatias = append(todasAsFatias,
apresentados...)
   todasAsFatias = append(todasAsFatias,
apresentando...)
   todasAsFatias = append(todasAsFatias,
apresentarao...)
   fmt.Println(todosOsSeminarios)
   fmt.Println(todasAsFatias)
   fmt.Println(seminarios)
   fmt.Println(apresentados)
   //fmt.Println(apresentados[6])
   fmt.Println(apresentando)
   fmt.Println(apresentarao)
}
```



Tipos Compostos - Slices - Exemplo 2

```
joaogcalen@LinuxDoRoberval ~/go/src/slices_and_append$ go
run slices_and_append.go
[Python C# TypeScript Scala Haskell Go Kotlin Lua Dart Visual Basic]
[Python C# TypeScript Scala Haskell Go Kotlin Lua Dart Visual Basic]
[Python C# TypeScript Scala Haskell Go]
[Python C# TypeScript Scala Haskell]
[Go]
[Kotlin Lua Dart Visual Basic]
```

Tipos Compostos - Maps

- Coleção não ordenada de pares chave-valor.
- Também chamado de array associativo, tabela hash ou dicionário
- Declaração: var mapa map[tipo1]tipo2
- Inicialização: mapa := make(map[tipo1]tipo2)
- Adicionar elementos: mapa[chave] = valor
- Remover elementos: delete(mapa, chave)
- map[int]tipo é diferente de array

Tipos Compostos - Maps - Exemplo

```go
package main

import "fmt"

var (
    mapa map[string]int
)

func main() {
    mapa["chave"] = 10
    fmt.Println(mapa["chave"])
}
```

```
joaogcalen@LinuxDoRoberval ~/go/src/teste$ go run teste.go
panic: assignment to entry in nil map

goroutine 1 [running]:
main.main()
        /home/joaogcalen/go/src/teste/teste.go:10 +0x52
exit status 2
```

Tipos Compostos - Maps - Correção Exemplo

```go
package main

import "fmt"

var (
    mapa map[string]int
)

func main() {
    mapa = make(map[string]int)
    mapa["chave"] = 10
    fmt.Println(mapa["chave"])
}
```

```
joaogcalen@LinuxDoRoberval ~/go/src/teste$ go run teste.go
10
```

Tipos Compostos - Maps - Exemplo 2

```go
package main
import "fmt"
func main() {
    elements := make(map[string]string)
    elements["H"] = "Hydrogen"
    elements["He"] = "Helium"
    elements["Li"] = "Lithium"
    elements["Be"] = "Beryllium"
    elements["B"] = "Boron"
    elements["C"] = "Carbon"
    elements["N"] = "Nitrogen"
    elements["O"] = "Oxygen"
    elements["F"] = "Fluorine"
    elements["Ne"] = "Neon"
    fmt.Println(elements["Li"])
    fmt.Println(elements["Un"])
}
```

Exemplo retirado do livro "an introduction to programming in go", de Caleb Doxsey

```
joaogcalen@LinuxDoRoberval ~/go/src/maps_chemistry$ go
run maps_chemistry.go
Lithium
```

Tipos Compostos - maps - Exemplo 3

```go
package main
import "fmt"
func main() {
   elements := map[string]map[string]string{
       "H": map[string]string{
           "name":  "Hydrogen",
           "state": "gas",
       },
       "He": map[string]string{
           "name":  "Helium",
           "state": "gas",
       },
       "Li": map[string]string{
           "name":  "Lithium",
           "state": "solid",
       },

   }
   if el, ok := elements["Li"]; ok {
       fmt.Println(el["name"], el["state"])
   }
}
```

```
joaogcalen@LinuxDoRoberval ~/go/src/maps_chemistry$ go
run maps_chemistry.go
Lithium solid
```

Tipos Compostos - Ponteiros

- Ponteiros são referências diretas para endereço de memória.
- Funciona de forma semelhante à C, porém sem aritmética
- Operador * para declaração de ponteiros.
- Operador & para retornar endereço de uma variável
- Função new funciona da mesma forma que *

```
var ponteiro *int
var ponteiro2 = new(int)
var valor int
valor = 5
ponteiro = &valor
ponteiro2 = &valor
```

Tipos Compostos - Structs

- Tipo definido com campos nomeados

```
type Circle struct {
   x,y,r float64
}
func (c *Circle) area() float64 {
   return math.Pi * c.r*c.r
}

var c1 Circle
c1 := new(Circle)
c2 := Circle{x: 0, y: 0, r: 5}
c3 := Circle{0, 0, 5}
fmt.Println(c1.x, c1.y, c1.r)
```

Variáveis e Constantes

Variáveis e Constantes - Declaração

Go possui inferência de tipo e as variáveis podem ser declaradas por:

- Declaração longa
  - `var i int`
  - `var i int = 1`
- Declaração curta
  - `i := 1`
  - `i := int64(1)`

Variáveis e Constantes - Valor Zero

Os tipos primitivos podem ser utilizados sem serem inicializados, pois cada tipo tem um valor padrão.

- `var a int // 0`
- `var b string // ""`
- `var c float64 // 0`
- `var d bool // false`
- `var e *int // nil`

Variáveis e Constantes - Nomes

Variáveis são case-sensitive e possuem as seguintes regras:

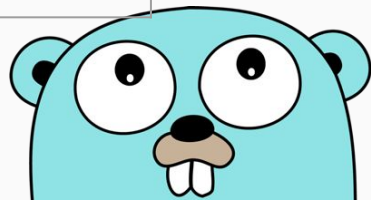
- Não podem conter espaços
- Não podem começar com número
- Só podem conter letras, números e underlines

Válido	Inválido
cachorroQuente	cachorro-quente
nome1	1nome
usuario	$usuario

Variáveis e Constantes - Acesso

A primeira letra de uma variável tem um significado especial.

`var Email string`	Pode ser acessada por outros pacotes
`var password string`	Só pode ser acessada dentro do pacote



Variáveis e Constantes - Atualização

Variáveis podem ser atualizadas desde que o novo valor seja do mesmo tipo.

```
a := 10
b := 11
a = 11
b = "12"
// cannot use "12" (type string) as type int in assignment
```

Variáveis e Constantes - Atribuição Múltipla

Variáveis e constantes permitem atribuição múltipla.

```
j, k, l := "j", 2, 2.5
const m, n, o = 1, 2, 3
```

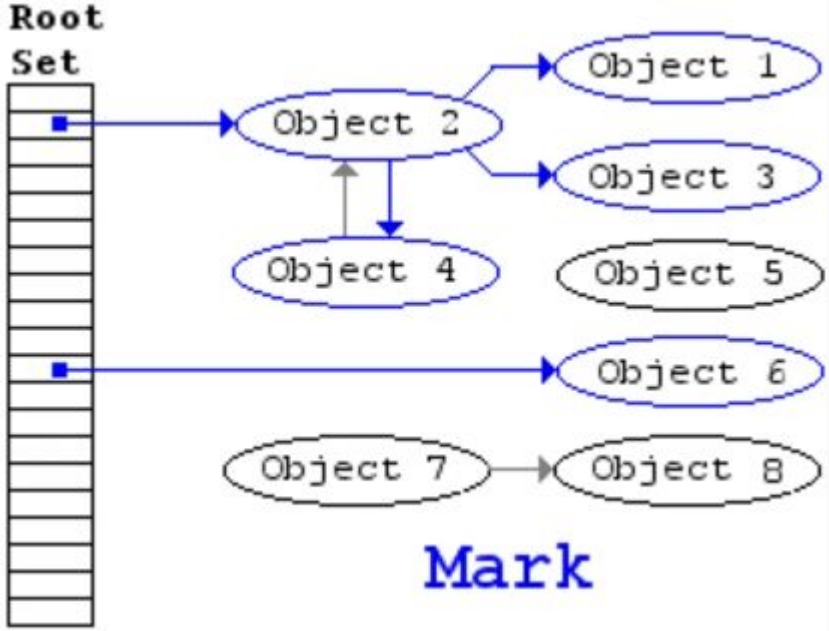
Variáveis e Constantes - Variáveis inutilizadas

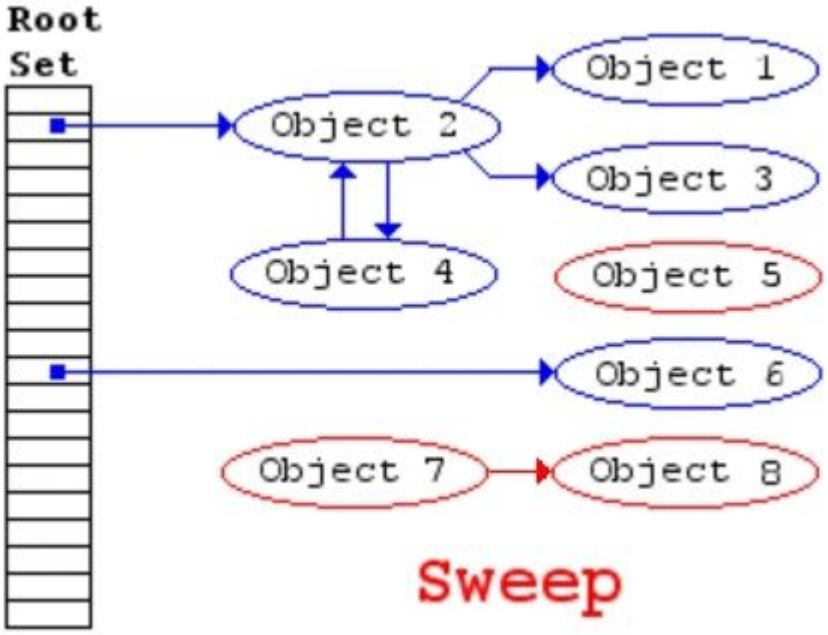
```
package main
import "fmt"
func main() {
    x := 5
    y := 1
    z := 2
    fmt.Println(y + z)
}//./prog.go:8:2: x declared and not used
```

Variáveis e Constantes - Coletor de Lixo

- Mesmo permitindo grande controle de memória pelo programador, Go possui coletor de lixo
- A implementação atual do coletor é um mark-and-sweep
- Em caso de múltiplos cores, o coletor roda em paralelo

Algoritmo mark-and-sweep





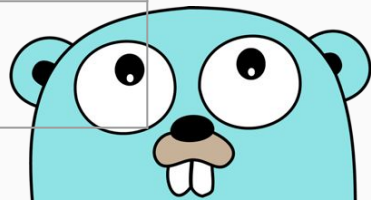
Comandos e expressões

Comandos e expressões - Operadores

- Operadores unários
- Operadores binários
- A sequência de tabelas de operadores binários nos próximos slides segue o nível de precedência adotado pela linguagem, de forma decrescente de prioridade.

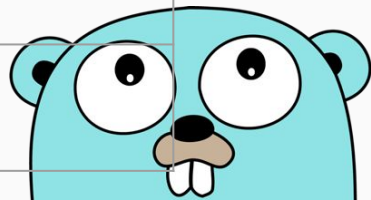
Comandos e expressões - Operadores unários

Operador	Descrição	Exemplo	Saída
*	Retorna o valor da variável apontada pelo ponteiro	y := 9 x := &y fmt.Println(*x)	9
&	Retorna o endereço da variável na memória	fmt.Println(&y)	0x549100
+	Dado x, retorna 0+x. Ou seja, não tem efeito.	fmt.Println(+y)	9
-	Retorna a negação do valor numérico	fmt.Println(-y)	-9



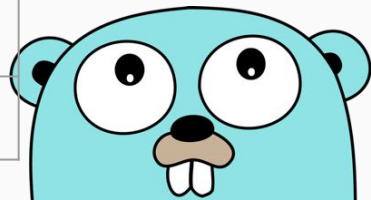
Comandos e expressões - Operadores unários

Operador	Descrição	Exemplo	Saída
++	Incremento em 1 unidade. Apenas posfixo.	x := 10 x++ fmt.Println(x)	11
--	Decremento em 1 unidade. Apenas posfixo.	x-- fmt.Println(x)	10
!	Negação de um booleano.	condicao := true fmt.Println(!condicao)	false
^	Negação de um número	fmt.Println(10)	-11



Comandos e expressões - Operadores binários - Nível 1

Operador	Descrição	Exemplo	Saída
*	Multiplicação	fmt.Println(10 * 2)	20
/	Divisão	fmt.Println(10 / 2)	5
%	Resto	fmt.Println(10 % 2)	0
<<	Shift Left	fmt.Println(1 << 4)	16
>>	Shift Right	fmt.Println(16 >> 4)	1
&	and bit a bit	fmt.Println(20 & 6)	4
&^	and not	fmt.Println(20 &^ 6)	16

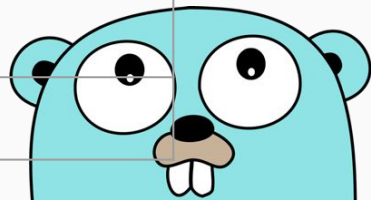


Comandos e expressões - Operadores binários - Nível 2

Operador	Descrição	Exemplo	Saída
+	Soma	fmt.Println(20 + 6)	26
-	Subtração	fmt.Println(20 - 6)	14
|	Ou (OR)	fmt.Println(20 | 6)	22
^	Ou exclusivo (XOR)	fmt.Println(20 ^ 6)	18

Comandos e expressões - Operadores binários - Nível 3

Operador	Descrição	Exemplo	Saída
==	Igual	x := 2 y := 2 fmt.Println(x == y)	true
!=	Diferente	fmt.Println(x != y)	false
<	Menor	fmt.Println(x < y)	false
<=	Menor igual	fmt.Println(x <= y)	true
>	Maior	fmt.Println(x > y)	false
>=	Maior igual	fmt.Println(x >= y)	true



Comandos e expressões - Operadores binários - Nível 4

Operador	Descrição
&&	e (AND) entre booleanos

Comandos e expressões - Operadores binários - Nível 5

Operador	Descrição
||	ou (OR) entre booleanos

Comandos e expressões

- If/else
- For
- Switch

Comandos e expressões - If / else

```
package main

func main() {
    if a := 7; a > 5 {
        // codigo
    } else {
        // codigo
    }
}
```

Comandos e expressões - For

```
sum := 0
for i := 0; i < 10; i++ {
sum += i
}

sum := 1
for ; sum < 1000; {
sum += sum
}
```

```
sum := 1
for sum < 1000 {
sum += sum
}

for{
   //loop infinito
}
```

Comandos e expressões - switch

```
nota := 0
switch nota {
case 0, 1, 1 + 1, 3:
    fmt.Println("Bom")
case 4, 5:
    fmt.Println("Excelente")
case 6, 7:
    fmt.Println("Um Deus da disciplina")
case 8, 9:
    fmt.Println("Com certeza colou")
case 10:
    fmt.Println("Impossível!")
default:
    fmt.Println(nota, " fora do limite")
}
```

Comandos e expressões - Curto circuito - Exemplo

```go
package main

import "fmt"

func main() {
    s := ""
    if s != "" && s[0] == 's' {
        fmt.Printf("Não é vazio, string: %s\n",
s)
    } else {
        fmt.Printf("É vazio, string: %s\n", s)
    }
}
```

```
joaogcalen@LinuxDoRoberval ~/go/src/curto_circuito$ go run
curto_circuito.go
É vazio, string:
```

Modularização

Pacotes

- Em go os programas são constituídos de pacotes.
- O pacote é definido utilizando a palavra chave “package”.
- O pacote main é o início do programa.
- O nome do pacote é o último elemento do caminho de importação

Pacotes

```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    fmt.Println("My favorite number is", rand.Intn(10))
}
```

Pacotes

- Existe dois modos de importação.
  - Consignada
  - Individual

```
//Consignada
import (
    "fmt"
    "math/rand"
)

//Individual
import "fmt"
import "math/rand"
```

Pacotes

- Uma função acessível de fora do pacote é dito como função exportada.
- Funções que começam com letra maiúscula são exportadas.

```
//Exportada
func Add(x int, y int){
    return x + y
}

//Não exportada
func sub(x int, y int){
    return x - y
}
```

Funções

- Funções são definidas pela palavra func.
- O tipo do parâmetro vem depois do nome.
- Parâmetros são passadas por cópia de valor.
- O retorno é o último elemento na declaração.
  - Caso não exista retorno, ele é assumido como void.

```
func add(x int, y int) int {
    return x + y
}
```

Funções

- Quando os parâmetros consecutivos possui o mesmo tipo, o tipo dos parâmetros anteriores ao último podem ser omitidos.

```
func add(x , y int) int {
    return x + y
}
```

Funções

- Uma função pode retornar múltiplos valores.

```
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    a, b := swap("hello", "world")
    fmt.Println(a, b)
}
```

Funções

- Os valores retornadas podem ser nomeadas e funcionarem como variáveis.
- Usar o return sem argumento implica no retorno das variáveis declaradas nos valores de retorno.

```
func split(sum int) (x, y int) {
    x = sum * 4 / 9
    y = sum - x
    return
}
```

Funções

- Funções podem receber outras funções como parâmetros

```
func compute(fn func(float64, float64) float64) float64 {
	return fn(3, 4)
}
func main() {
	hypot := func(x, y float64) float64 {
		return math.Sqrt(x*x + y*y)
	}
	fmt.Println(hypot(5, 12))

	fmt.Println(compute(hypot))
	fmt.Println(compute(math.Pow))
}
```

Funções

- Também há as funções closures

```
func op(fn func(a, b int) int, x, y int) int{
	return fn(x, y)
}
func main() {
	for i := 0; i < 10; i++ {
		fmt.Println(
			op(func(a, b int) int{
				return a * b
			}, i, 2),
		)
	}
}
```

Polimorfismo

Métodos

- Go não possui classes, porém podemos definir métodos que operam sobre tipos.
- Um método é uma função que possui um receptor especial.

```
//Com receptor
func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
Sem receptor
func Abs(v Vertex) float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

Métodos

- A chamada do método é similar à chamada de um método de classe.

```
func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func main() {
    v := Vertex{3, 4}
    fmt.Println(v.Abs())
}
```

Métodos

- Os métodos podem ter como receptores somente os tipos que estão no mesmo pacote, tipos de outros pacotes não são permitidos e isso inclui os tipos do pacote padrão, ex: int.
- Os receptores podem ser ponteiros (note que não é necessário a sintaxe “(&v).X” para acessar o valor da variável, go faz isso automaticamente.)

```
func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
```

Interfaces

- Interface é um tipo que define um conjunto de métodos.

```
type Abser interface {
    Abs() float64
}
```

Interfaces

- Um valor é do tipo da interface quando ele implementa todos os métodos.
- As implementações das interfaces são implícitas, ou seja, não é necessário explicitar a intenção de implementar uma interface.

```
type Abser interface {
    Abs() float64
}
type Vertex struct {
    X, Y float64
}
func (v *Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

Interfaces

- Como Vertex implementa abs(), ele pode ser atribuído a uma variável do tipo da interface Abser.

```
func main() {
    var a Abser
    v := Vertex{3, 4}

    a = &v // a *Vertex implements Abser

    fmt.Println(a.Abs())
}
```

Interfaces

- Podemos acessar o valor concreto da interface utilizando Type assertation

```
func main() {
    var i interface{} = "hello"

    s := i.(string)
    fmt.Println(s)

    f := i.(float64) // erro
    fmt.Println(f)
}
```

Interfaces

```
type doisInt struct{
    a int
    b int
}

type I interface{
    aa() int
}

func (b doisInt) aa() int{
    return b.a
}
```

```
func main() {
    var x = doisInt{1, 2}

    var i I
    i = x

    fmt.Println(i.(doisInt).a)

}
```

Interface

- Podemos também acessar o valor da interface com type swicth.
- O variável “v” irá assumir o valor do tipo que está explícito no case, caso ela seja realmente daquele tipo. Se cair no case default ela continua sendo do tipo da interface.

```
switch v := i.(type) {
    case int:
        fmt.Printf("Twice %v is %v\n", v, v*2)
    case string:
        fmt.Printf("%q is %v bytes long\n", v, len(v))
    default:
        fmt.Printf("I don't know about type %T!\n", v)
    }
```

Exceções

Exceções

Não há exceções. Existem duas maneiras de tratamento de erro:

- Usar múltiplos valores de retorno
- Defer, Panic e Recover

Exceções - Interface error

Por convenção, erros são criados pela interface padrão error. O programador deve implementar essa interface para que seja possível determinar o contexto do erro.

```
type error interface {
    Error() string
}
```

Exceções - Implementação da Interface Error

Exemplo de erro da função os.Open()

```
// PathError records an error and the operation and
// file path that caused it.
type PathError struct {
    Op string    // "open", "unlink", etc.
    Path string  // The associated file.
    Err error    // Returned by the system call.
}

func (e *PathError) Error() string {
    return e.Op + " " + e.Path + ": " +
e.Err.Error()
}

//open /etc/passwx: no such file or directory
```

Exceções - Múltiplos Valores de Retorno

Erros são comunicados por um valor de retorno separado.

```
import (
    "errors"
    "fmt"
)
func f1(arg int) (int, error) {
    if arg == 42 {

        return -1, errors.New("não funciona com 42")

    }

    return arg + 3, nil
}
func main() {
    for _, i := range []int{7, 42} {
        if r, e := f1(i); e != nil {
            fmt.Println("f1 falhou:", e)
        } else {
            fmt.Println("f1 funcionou:", r)
        }
    }
}
```

```
//Output:
//f1 funcionou: 10
//f1 falhou: não funciona com 42
```

Exceções - Múltiplos Valores de Retorno

Implementando a interface para o exemplo anterior

```
func main() {
    for _, i := range []int{7, 42} {
        if r, e := f2(i); e != nil {
            fmt.Println("f2 falhou:", e)
        } else {
            fmt.Println("f2 funcionou:", r)
        }
    }

    _, e := f2(42)
    if ae, ok := e.(*argError); ok {
        fmt.Println(ae.arg)
        fmt.Println(ae.prob)
    }
}
//f2 funcionou: 10
//f2 falhou: 42 - não funciona
//42
//não funciona
```

```
type argError struct {
    arg  int
    prob string
}

func (e *argError) Error() string {
    return fmt.Sprintf("%d - %s", e.arg, e.prob)
}

func f2(arg int) (int, error) {
    if arg == 42 {

        return -1, &argError{arg, "não funciona"}
    }
    return arg + 3, nil
}
```

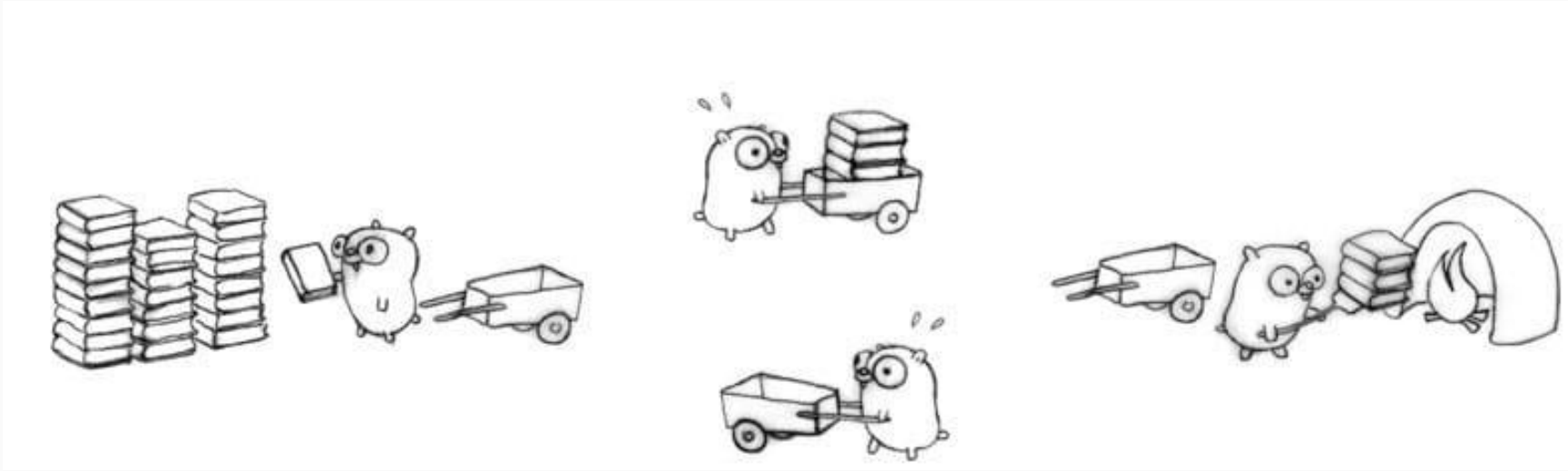
Exceções - Defer, Panic e Recover

- Defer: empilha uma chamada de função. Funções empilhadas são sempre executadas imediatamente antes do final da função onde foram chamadas, em ordem LIFO. Pode ser usado como um finally.
- Panic: interrompe a função atual com uma mensagem de erro e as funções deferidas são executadas. Pode ser usado como um throw.
- Recover: obtém a mensagem de erro gerada por panic e a retorna. Pode ser usado como um catch.

Exceções - Defer, Panic e Recover como Try, Catch e Finally

```
func tratador(){
    if mensagemPanico := recover(); mensagemPanico != nil{
        fmt.Println(mensagemPanico)
    }
}

func area(lado int) (int){
    defer tratador()

    if lado < 0 {
        panic("Panic: Lado negativo!")
    }
    return lado * lado
}

func main() {                          //Panic: Lado negativo
    i := area(-5)                      //i = 0
    fmt.Printf("i = %d", i)

}
```

Concorrência



Concorrência

- Go tem suporte parcial para concorrência integrado na linguagem
- Usa-se a palavra chave go para iniciar uma thread
- Possui um pacote “sync” que contém primitivas funções e tipos para concorrência
- Uma thread em Go é chamada Goroutine
- Não se tem suporte de prioridades, são cuidadas pela linguagem
- Go 1.5 ou mais novo consegue usar mais de um núcleo do processador quando disponível

Concorrência

Além do pacote “sync”, o pacote “time” será usado para exemplos de execução pesada. “time” possui funcionalidade para medir e mostrar tempo.

```
import (
    "fmt"
    "time"
    "sync"
)
```

Concorrência

```
func say() {
    // executa na Goroutine que a main() cria
    fmt.Print("Hello")
}

func main() {
    go say() // cria uma Goroutine
    // Faz uma tarefa que dura um tempo, aqui 1 segundo
    time.Sleep(time.Duration(1) * time.Second)
    fmt.Println(", World!")
}
```

Concorrência

Uma das saídas do programa anterior é:

Hello, World!

Mas que também pode ser:

, World!Hello

Concorrência, sync.WaitGroup

- Como esperar que duas ou mais goroutines acabem? WaitGroup
- Crie um grupo com tipo sync.WaitGroup: `var wg sync.WaitGroup`
- Adicione cada goroutine para o grupo: `wg.Add(n)`
- Para cada goroutine terminada, avise ao grupo que terminou: `wg.Done()`
- Espere todas as goroutines acabarem: `wg.Wait()`

```go
// O programa cria duas goroutines que imprime "A" e "B" concorrentemente.
// A saída está na direita do código abaixo. Entre cada "A" consecutivo,
// possui um tempo de 20 milissegundos. O mesmo para "B". Enquanto isso,
// main() espera ambas as rotinas acabarem
func show(name string, wg *sync.WaitGroup) {
    for i := 1; i <= 5; i++ {
        time.Sleep(time.Duration(20) * time.Millisecond)
        fmt.Println(name, i)
    }
    wg.Done() // avisando que acabou
}
func main() {
    var wg sync.WaitGroup
    wg.Add(2) // terão duas goroutines para esperar
    go show("A", &wg)
    go show("B", &wg)
    wg.Wait() // espere todas as goroutines acabarem
}
```

Saída

```
1.  A 1
2.  B 1
3.  B 2
4.  A 2
5.  A 3
6.  B 3
7.  B 4
8.  A 4
9.  A 5
10. B 5
```

Concorrência, sincronização

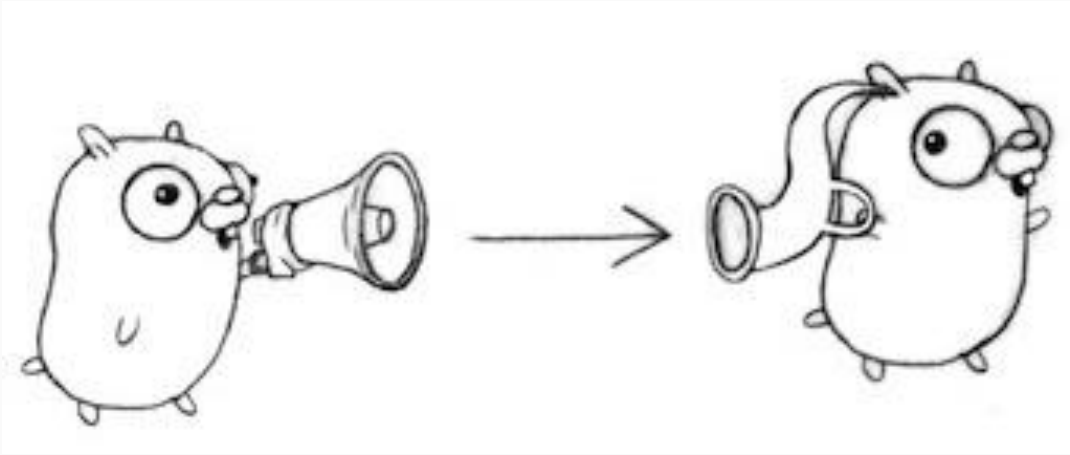
- Go possui sincronização com semáforos e não possui estrutura para isso
- Go possui uma primitiva em sync para exclusão mútua: `sync.Mutex`
- Proberen (testar): `sync.Mutex.Lock()`
- Verhogen (incrementar): `sync.Mutex.Unlock()`

Concorrência, semáforos

```go
// Em Go
var m sync.Mutex

// goroutine 1
m.Lock()
// faça algo
m.Unlock()

// goroutine 2
m.Lock()
// faça algo
m.Unlock()
```

```java
// Em Java
Object lock = new Object();

// thread 1
synchronized (lock) {
    // faça algo
}

// thread 2
synchronized (lock) {
    // faça algo
}
```

Concorrência, canais.



Concorrência, canais.

```
func main() {
    var c chan int = make(chan int)
    c <- 10 // envia
    x := <-c // recebe
    fmt.Println(x)
}
```

Concorrência, canais.

```
func sender(onii chan int) {
	for i := 0; i < 3; i++ {
		onii <- i
	}
	close(onii)
}
func main() {
	c := make(chan int)
	go sender(c)
	for {
		v, ok := <-c
		if !ok { break }
		fmt.Println(v)
	}
}
```

Concorrência, canais.

```
for {
    v, ok := <-c
    if !ok { break }
    fmt.Println(v)
}


for v := range c {
    fmt.Println(v)
}
```

Concorrência, canais.

```
func sender(onii chan<- string, name string, t int) {
	for {
		time.Sleep(time.Millisecond * time.Duration(t))
		onii <- name
	}
	close(onii)
}
func main() {
	a := make(chan string)
	b := make(chan string)
	go sender(a, "A", 100)
	go sender(b, "B", 300)
	for {
		fmt.Println(<-a)
		fmt.Println(<-b)
	}
}
```


Concorrência, canais.

```
select {
case v := <- a:
    fmt.Println(v)
case v := <- b:
    fmt.Println(v)
default:
    fmt.Println("Nenhum")
}
```

Concorrência, canais.

```
func remetente(c chan<- string)

func destinatario(c <-chan string)

invalid operation: onii <- name (send to receive-only type
<-chan string)
invalid operation: close(onii) (cannot close receive-only
channel)
```

Comparação entre C, C++, Java e Go

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Sim
Portabilidade	Não	Não	Sim	Sim
Método de projeto	Estruturado	Estruturado e OO	OO	Estruturado e Concorrente
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Critérios gerais	C	C++	Java	Go
Aplicabilidade	**Sim**	**Sim**	**Parcial**	**Sim**
Confiabilidade	Não			
Aprendizado	Não			
Eficiência	Sim			
Portabilidade	Não	Não	Sim	Sim
Método de projeto	Estruturado	Estruturado e OO	OO	Estruturado e Concorrente
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Propósito geral

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	**Não**	**Não**	**Sim**	**Sim**
Aprendizado				
Eficiência				
Portabilidade				
Método de projeto	Estruturado	Estruturado e OO	OO	Estruturado e Concorrente
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Checagem de índice, segmentation violation, coleta de lixo.
Pacote “unsafe” com funções inseguras.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	**Não**	**Não**	**Não**	**Não**
Eficiência	S			
Portabilidade				
Método de projeto				
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Assim como C, Go também tem o conceito de ponteiros.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	**Sim**	**Sim**	**Parcial**	**Sim**
Portabilidade	N			
Método de projeto	E			
Evolutibilidade	Não			
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Go compila o código para linguagem de máquina assim como C e C++, para a arquitetura atual. Isso faz com que Go seja rápido assim como C.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Sim
Portabilidade	**Não**	**Não**	**Sim**	**Sim**
Método de projeto	E			o e e
Evolutibilidade	N			
Reusabilidade	F			
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

O código em Go não precisa ser reescrito ou adaptado para outra arquitetura ou plataforma, apenas os arquivos construídos após a compilação. Go consegue gerar executáveis para várias plataformas. Já que executáveis são dependentes da plataforma, também pode considerar que Go tem portabilidade parcial.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	N			
Aprendizado	N			
Eficiência	S			
Portabilidade	N			
Método de projeto	**Estruturado**	**Estruturado e OO**	**OO**	**Estruturado, OO e Concorrente**
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Go tem um método de concorrência usando goroutines. Go também adota OO parcialmente, pois não possui hierarquia de objetos, mas possui structs que se comportam como objetos.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim		Parcial	Sim
Portabilidade	Não		Sim	Sim
Método de projeto	Estru		OO	Estruturado e Concorrente
Evolutibilidade	**Não**	**Parcial**	**Sim**	**Sim**
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Go usa OO, com interfaces implícitas.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Sim
Portabilidade	N			
Método de projeto	E			ado e ente
Evolutibilidade	Não			
Reusabilidade	**Parcial**	**Sim**	**Sim**	**Sim**
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Go permite o reuso de tipos definidos, funções e pacotes.
Go possui vários pacotes com foco em serviços de internet.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Sim
Portabilidade	Nã			
Método de projeto	E			
Evolutibilidade	N			
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	**Sim**	**Sim**	**Parcial**	**Parcial**
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Gratuito

Go consegue integrar com a linguagem C com um import “C” acima um comentário contendo o código em C.

Critérios gerais	C	C++	Java	Go
Aplicabilidade	Sim	Sim	Parcial	Sim
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Sim
Portabilidade	Não	Não	Sim	Sim
Método de projeto	Estruturado	Estruturado e OO	OO	Estruturado e rrente
Evolutibilidade	N			
Reusabilidade	F			
Integração	Si			al
Custo	**Depende da ferramenta**	**Depende da ferramenta**	**Depende da ferramenta**	**Gratuito**

A linguagem e o código fonte são gratuitos.
https://golang.org/PATENTS

Critérios específicos	C	C++	Java	Go
Escopo	Sim	Sim	Sim	Parcial
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	OpenDB, serialização, biblioteca de tipos e funções
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável e por valor

Critérios específicos	C	C++	Java	Go
Escopo	**Sim**	**Sim**	**Sim**	**Sim**
Expressões e comandos	Definição explícita de entidades			
Tipos primitivos e compostos				
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	OpenDB, serialização, biblioteca de tipos e funções
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável e por valor

Critérios específicos	C	C++	Java	Go
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	**Sim**	**Sim**	**Sim**	**Sim**
Tipos primitivos e compostos				
Gerenciamento de memória				
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	OpenDB, serialização, biblioteca de tipos e funções
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável e por valor

Go oferece uma grande variedade de comandos, e muitos são focados para serviços em internet.

Critérios específicos	C	C++	Java	Go
Escopo	Sim	Sim	Sim	Parcial
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	**Sim**	**Sim**	**Sim**	**Sim**
Gerenciamento de memória				
Persistência dos dados			serialização	o, de tipos e funções
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável e por valor

Go tem vários tipos primitivos em que alguns possuem tamanho fixo (ex. int32) e também podem ter tamanho definido pela arquitetura (ex. int)

Critérios específicos	C	C++	Java	Go
Escopo	Sim	Sim	Sim	Parcial
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	**Programador**	**Programador**	**Sistema**	**Sistema**
Persistência dos dados				o, de ões
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável e por valor

Go tem coletor de lixo

Critérios específicos	C	C++	Java	Go
Escopo	Sim	Sim	Sim	Parcial
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos				
Gerenciamento de memória				a
Persistência dos dados	**Biblioteca de funções**	**Biblioteca de classes e funções**	**JDBC, biblioteca de classes, serialização**	**OpenDB, serialização, biblioteca de tipos e funções**
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável e por valor

Go tem serialização de objetos, usando os pacotes `"encoding/base64"` `"encoding/gob"` `"bytes"` , leitura e escrita de arquivos e pacotes de interface com banco de dados usando drivers. Drivers suportados: https://github.com/golang/go/wiki/SQLDrivers

Critérios específicos	C	C++	Java	Go
Escopo	Sim	Sim	Sim	Parcial
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória				
Persistência dos dados		funções	serialização	tipos e funções
Passagem de parâmetros	**Lista variável e por valor**	**Lista variável, default, por valor e por referência**	**Lista variável, por valor e por cópia de referência**	**Lista variável e por valor**

Assim como C, passagem é por valor assim precisando usar ponteiros.

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Estática	Estática / Dinâmica	Estática / Dinâmica	Estática
Polimorfismo	Coerção e sobrecarga	Todos	Todos	Coerção, sobrecarga e Inclusão (upcast)
Exceções	Não	Parcial	Sim	Não
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	**Parcial**	**Sim**	**Sim**	**Sim**
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Está			
Polimorfismo	Coerção e sobrecarga	Todos	Todos	sobrecarga e Inclusão (upcast)
Exceções	Não	Parcial	Sim	Não
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Go esconde entidades quando a primeira letra é minúscula e exporta entidades quando maiúsculas.

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	**Não**	**Parcial**	**Sim**	**Sim**
Verificação de tipos				ca
Polimorfismo				ão, carga e Inclusão (upcast)
Exceções	Não	Parcial	Sim	Não
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Go possui ponteiros, mas não permite coerção entre eles, não permite aritmética (não numérico).

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	**Estática**	**Estática / Dinâmica**	**Estática / Dinâmica**	**Estática**
Polimorfismo				ão, carga e ão (upcast)
Exceções				
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Verificação de tipos são feitos em compilação.

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Nã			
Verificação de tipos	Es	Dinâmica	Dinâmica	
Polimorfismo	**Coerção e sobrecarga**	**Todos**	**Todos**	**Coerção, sobrecarga e Inclusão (upcast)**
Exceções	Não	Parcial	Sim	Não
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Com uso de interfaces, é possível ter polimorfismo.

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Es			
Polimorfismo	Co sobrecarga			rga e Inclusão (upcast)
Exceções	**Não**	**Parcial**	**Sim**	**Não**
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Go não possui sistema de exceção.

Critérios específicos	C	C++	Java	Go
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Estática	Estática / Dinâmica	Estática / Dinâmica	Estática
Polimorfismo	C s			ho, arga e o (upcast)
Exceções	Não	Parcial	Sim	Não
Concorrência	**Não (biblioteca de funções)**	**Não (biblioteca de funções)**	**Sim**	**Sim**

Go oferece recursos nativos para criar goroutines, channels para comunicação entre as goroutines e possui pacotes para primitivas em goroutines como “sync”.

https://stackoverflow.com/questions/31353522/why-must-i-convert-an-integer-to-a-float64-to-type-match
https://yourbasic.org/golang/split-string-into-slice/
https://yourbasic.org/golang/structs-explained/
https://stackoverflow.com/questions/9320862/why-would-i-make-or-new
https://golang.org/pkg/strconv/
https://gobyexample.com/command-line-arguments
https://stackoverflow.com/questions/14426366/what-is-an-idiomatic-way-of-representing-enums-in-go
https://spf13.com/post/is-go-object-oriented/
https://yourbasic.org/golang/for-loop-range-array-slice-map-channel/
https://gobyexample.com/writing-files
https://tour.golang.org/flowcontrol/12
https://stackoverflow.com/questions/1821811/how-to-read-write-from-to-file-using-go
https://stackoverflow.com/questions/35333302/how-to-write-the-output-of-this-statement-into-a-file-in-golang
https://stackoverflow.com/questions/25928991/go-print-without-space-between-items
https://stackoverflow.com/questions/48111385/how-to-print-float-number-where-decimal-point-is-comma-in-fmt-sprintf
https://godoc.org/golang.org/x/text/message
https://yourbasic.org/golang/round-float-2-decimal-places/
http://tleyden.github.io/blog/2014/10/30/goroutines-vs-threads/

https://www.youtube.com/watch?v=C8LgvuEBraI
https://stackoverflow.com/questions/28799110/how-to-join-a-slice-of-strings-into-a-single-string
https://golang.org/pkg/sort/
https://golang.org/ref/spec#Rune_literals
https://golang.org/ref/spec#Keywords
https://www.slant.co/versus/113/126/~c_vs_go
https://www.quora.com/What-is-golang-good-for
https://github.com/google/periph
https://github.com/google/gousb
https://golang.org/doc/faq#garbage_collection
https://blog.golang.org/ismmkeynote
https://pt.stackoverflow.com/questions/13416/como-garbage-collection-%C3%A9-implementado-em-go
https://stackoverflow.com/questions/23632072/why-is-this-code-undefined-behavior
https://stackoverflow.com/questions/50830676/set-int-pointer-to-int-value-golang
https://stackoverflow.com/questions/41220586/how-can-i-convert-pointer-type-in-golang
https://groups.google.com/forum/#!topic/golang-nuts/MB1QmhDd_Rk
https://www.youtube.com/user/PewDiePie/
https://golang.org/doc/faq
https://golang.org/doc/articles/race_detector.html
https://stackoverflow.com/questions/20240179/nil-detection-in-go

https://tour.golang.org/moretypes/7
https://willowtreeapps.com/ideas/the-pros-and-cons-of-programming-in-go
https://www.reddit.com/r/golang/comments/506ybq/how_portable_are_go_binaries/
https://stackoverflow.com/questions/1713214/how-to-use-c-in-go
https://stackoverflow.com/questions/28020070/golang-serialize-and-deserialize-back
https://github.com/golang/go/wiki/SQLDrivers
https://golang.org/pkg/database/sql/
https://softwareengineering.stackexchange.com/questions/77436/is-googles-go-a-type-safe-language
https://stackoverflow.com/questions/19612449/default-value-in-gos-method
https://stackoverflow.com/questions/5367961/casting-from-one-pointer-to-pointer-type-to-another-in-golang-error
https://stackoverflow.com/q/6986944/3491102
http://tleyden.github.io/blog/2014/10/30/goroutines-vs-threads/
DONOVAN,A.;KERNIGHAN,B. The Go Progamming Language: Crawfordsville, 2015.
DOXSEY, C.An Introduction to Programming in Go: 2012
AIMONETTI, M. Go Bootcamp