É Simples, mas Não É Fácil

ARTHUR SOUZA SEPP

Introdução

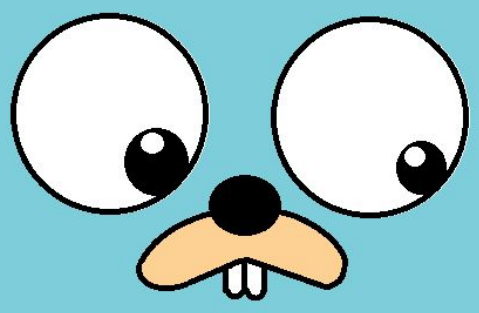
Aprender uma nova linguagem de programação pode parecer desafiador — mas o Go (ou Golang) foi criado justamente para simplificar o que costuma ser complicado.

Desenvolvido pelo Google, o Go combina performance de linguagens compiladas com a facilidade das linguagens modernas.

Neste e-book, você vai entender como essa linguagem, de aparência simples, esconde uma filosofia poderosa: clareza, velocidade e eficiência.

A cada capítulo, você aprenderá na prática como usar variáveis, funções e pacotes — os blocos fundamentais de qualquer aplicação Go — com exemplos reais e diretos ao ponto.

Prepare-se para descobrir que “simples” não significa “fácil”, mas que o Go foi feito para tornar o aprendizado leve, lógico e divertido.



01
VARIÁVEIS

Tudo em programação começa com dados. Neste capítulo, vamos aprender como o Go lida com variáveis e tipos de dados - desde números e textos até valores lógicos

Tipos e variáveis

Entendendo os dados

Em Go, tudo começa com os dados.
Antes de criar funções ou pacotes, é essencial entender como declarar e usar variáveis.

Go é uma linguagem **fortemente tipada,** o que significa que cada variável tem um tipo definido - e o compilador garante que você não misture tipos de maneira incorreta.

Os principais são:

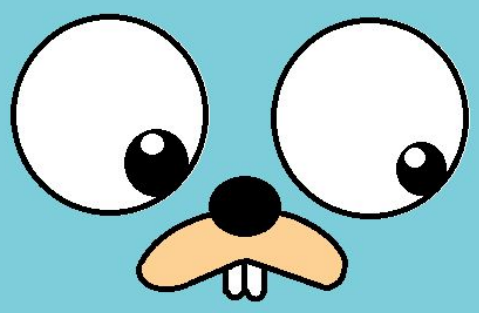
Tipo	Exemplo	Descrição
int	`10`	Números inteiros
float64	`3.14`	Números decimais
string	`“Olá”`	Texto
bool	`true ou false`	Valores lógicos

Exemplo

Usando variáveis para exibir dados:

```
package main

import “fmt”

func main() {
     var idade int = 25
     var nome string = “Manoel”
     saldo := 1520.75 // Definindo o tipo automaticamente

     fmt.Printf(“%s tem %d anos e saldo de R$ %.2f\n", nome,
     idade, saldo)
}
```

- Se você usar o operador `:=` , o Go descobre automaticamente qual o tipo da variável. Isso deixa o código mais limpo



02

TIPOS COMPOSTOS

Quando o programa começa a crescer, é preciso organizar melhor os dados. Vamos aprender a usar slices e structs para representar listas informações do mundo real — como produtos, usuários e pedidos — de forma clara e eficiente.

Estruturando informações

Organize seus dados

Além dos tipos básicos, Go oferece **tipos compostos**, perfeitos para representar coleções ou estruturas mais complexas:

- Slices - listas flexíveis

```
nomes := []string{“Ana”, “Bruno”, “Carlos”}
nomes = append(nomes, “Diana”) // Adicionando Diana à lista

fmt.Println(nome)
```

Slices são como listas dinâmicas - você pode adicionar elementos a elas com o *append*.

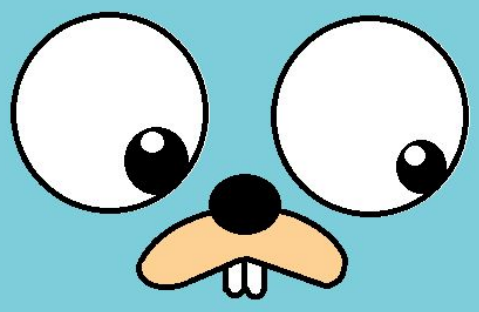
Estruturando informações

Organize seus dados

- Structs - modelos de dados

```
type Produto struct {
      Nome  string
      Preco float64
      Estoque int
}

func main() {
      p := Produto{“Mouse”, 89.90, 20} // 20 Mouses de $ 89
      fmt.Printf("Produto: %s | R$%.2f | Estoque: %d\n",
      p.Nome, p.Preco, p.Estoque)
}
```

As *structs* são como “modelos de objetos” em outras linguagens. Use-as para representar entidades do mundo real, como produtos, usuários, pedidos, entre outros..



03

FUNÇÕES

Funções são o coração do Go.
Elas permitem dividir o código em partes pequenas, reutilizáveis e fáceis de entender. Aqui, veremos como declarar funções, passar parâmetros, retornar resultados e até lidar com erros.

Minimize seu código

Funções incrementam sua organização

Funções são blocos de código que executam tarefas específicas e ajudam a manter seu programa organizado e reutilizável

A estrutura de uma função:

```
func nomeFuncao(parametro tipo, parametro2 tipo) tipo retorno
{
    // Código aqui
}
```

Exemplo prático:

```
func CalcularDesconto(preco float64, desconto float64) float64
{
     valorFinal := preco - (preco * desconto / 100)
     return valorFinal
}

func main() {
    precoComDesconto := CalcularDesconto(100.0, 15)
    fmt.Println(“Preço final:“, precoComDesconto)
}
```

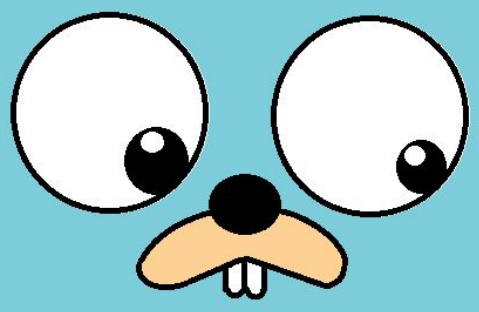
Aqui, *CalcularDesconto* recebe o preço e o percentual de desconto, retornando o valor final.

- **Dica**

Go permite funções retornarem múltiplos valores

```
func Dividir(a, b float64) (float64, error) {
     if b == 0 {
          return 0, fmt.Errorf(“não é possível dividir por 0”)
     }
     return a / b, nil
}
```

Essa abordagem torna o código mais seguro e expressivo, além de ser ótima para tratar erros.



04

PACOTES

Programas bem feitos são compostos por partes bem organizadas.
Aqui, veremos como pacotes ajudam a separar responsabilidades, reaproveitar código e deixar o projeto limpo.

Organização

Pacotes ajudam a organizar seu código

Agora que você entende **variáveis** e **funções**, é hora de aprender a **organizar seu código**. Em Go, isso é feito com **pacotes (packages).**

Por exemplo, o arquivo principal do seu programa geralmente usa:

```
package main

import “fmt”

func main() {
    fmt.Println(“Hello GO!”)
}
```

Aqui:

- *package main* indica que este arquivo é o ponto de entrada do programa
- *import fmt* traz um pacote de biblioteca padrão, usado para formatar e imprimir textos

Criando seus pacotes

Para casos mais específicos, você também pode criar seu próprio pacote:

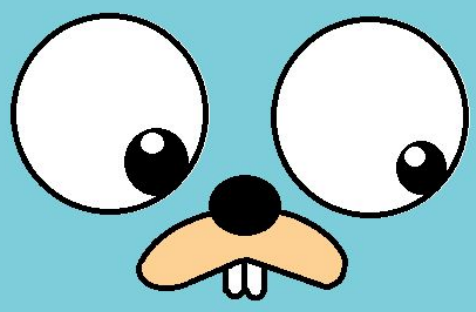
```go
// Arquivo: mathutils/soma.go:

package mathutils

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

E usá-lo no código principal:

```go
package main
import (
	“fmt”
	“meuprojeto/mathutils”
)

func main() {
	resultado := mathutils.Somar(10, 5)
	fmt.Println(“Soma:”, resultado)
}
```

Dicas:

- Pacotes tornam o código **modular e reutilizável.**
- Cada parte do sistema pode ficar em seu próprio pacote - mais fácil de manter e evoluir



05

JUNTANDO TUDO

Agora é hora de juntar tudo o que aprendemos. Neste capítulo final, veremos um pequeno projeto integrando tipos, funções e pacotes, entendendo como esses elementos se conectam para formar aplicações reais e bem estruturadas.

Conectando conceitos

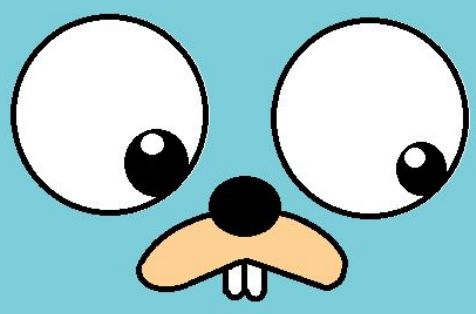
Vamos juntar tudo que vimos até então

Arquivo loja/produto.go:

```
package loja

type Produto struct {
     Nome      string
     Preco     float64
}

func (p Produto) PrecoComDesconto(desconto float64) float64
{
     return p.Preco - (p.Preco * desconto / 100)
}
```

Arquivo principal - main.go:

```
package main

import (
     “fmt”
     “meuprojeto/loja”
)

func main() {
     produto := loja.Produto{“Camiseta”, 59.90}
     fmt.Printf(“Preço com desconto: $%.2f\n”,
     produto.PrecoComDesconto(10))
}
```



CONCLUSAO

Obrigado por ler até aqui!

O conteúdo deste e-book foi totalmente gerado por Inteligência Artificial com fins de estudo e pesquisa. Podem conter erros gerados por um modelo de I.A

Os arquivos deste projeto podem ser encontrados no meu GitHub.

https://www.github.com/arthursepp/