# Introdução ao Go

## Aula 8: Trabalhando com Variáveis, Valores e Operadores

Nesta aula, o professor começa a ensinar **como declarar variáveis, armazenar valores e realizar cálculos simples em Go**. O foco é entender os conceitos de **variáveis, operadores matemáticos e tipos de dados**.

---

### Declaração de Variáveis

Em Go, as variáveis são declaradas com a palavra-chave `var`. A sintaxe básica é:

```go
var nomeVariavel tipo = valor
```

Se o tipo não for especificado, ele será inferido automaticamente com base no valor atribuído. Um exemplo prático:

```go
var investmentAmount = 1000
var expectedReturnRate = 5.5
var years = 10
```

- `investmentAmount`: Armazena o valor inicial do investimento.
- `expectedReturnRate`: Armazena a taxa de retorno esperada (em percentual).
- `years`: Número de anos para o investimento.

### Convenção de Nomes

A convenção de nomenclatura mais usada em Go é o **camelCase**, onde a primeira letra é minúscula e cada palavra subsequente começa com letra maiúscula (exemplo: `investmentAmount`).

---

### Fórmula do Cálculo do Valor Futuro

O cálculo do valor futuro é baseado na fórmula:

**FV = Principal × (1 + Taxa)$^t$**

Aqui:

- `Principal`: Valor inicial do investimento.
- `Taxa`: Taxa de retorno anual (em forma decimal, como 0.055 para 5.5%).
- `t`: Número de anos do investimento.

Em Go, isso pode ser representado assim:

```go
var futureValue = investmentAmount * (1 + (expectedReturnRate / 100))
```

---

### Precedência de Operadores e Parênteses

Os operadores em Go seguem as mesmas regras de precedência matemática que você usa no papel. Para garantir que a operação seja executada corretamente, é necessário usar parênteses. Por exemplo:

```go
futureValue := investmentAmount * (1 + (expectedReturnRate / 100))
```

---

### Problema de Tipos

Ao tentar realizar operações matemáticas em Go, você pode se deparar com um **erro de tipos**. Isso acontece porque Go é uma linguagem de **tipagem estática e forte**, ou seja:

- Uma variável de um tipo (ex.: `int`) não pode ser usada diretamente em operações com variáveis de outro tipo (ex.: `float64`).

No exemplo da aula:

- `investmentAmount` foi armazenado como um **inteiro** ( `int` ).
- `expectedReturnRate` foi armazenado como um **número decimal** ( `float64` ).
- A operação matemática mistura tipos diferentes ( `int` e `float64` ), o que gera o erro.

---

### Solução: Conversão de Tipos

Para resolver o problema de incompatibilidade de tipos, podemos converter `investmentAmount` para o tipo `float64`. A conversão é feita assim:

```go
futureValue := float64(investmentAmount) * (1 + (expectedReturnRate / 100))
```

Com isso, todas as variáveis usadas na operação têm o mesmo tipo ( `float64` ), e o erro desaparece.

---

## Código Completo da Aula

```go
package main

import "fmt"

func main() {
    var investmentAmount = 1000         // Valor inicial do investimento (int)
    var expectedReturnRate = 5.5        // Taxa de retorno anual esperada (float64)
    var years = 10                      // Tempo de investimento em anos (int)

    // Cálculo do valor futuro (conversão de tipos necessária)
    var futureValue = float64(investmentAmount) * (1 + (expectedReturnRate / 100))

    // Exibição do resultado
    fmt.Println("O valor futuro do investimento é:", futureValue)
}
```

**Pontos Importantes:**

1. Variáveis tornam o código mais legível e fácil de alterar.
2. O uso correto de tipos ( `int` , `float64` , etc.) é essencial para evitar erros.
3. A conversão de tipos ( `float64(var)` ) é necessária em operações que misturam tipos diferentes.

Se precisar de mais detalhes ou dúvidas sobre algum trecho, é só falar!