Introdução ao Go

Aula 33: Repetindo Código com Loops "for"

Nesta aula, o professor introduziu o uso do loop `for` em Go para repetir um bloco de código várias vezes até que uma condição específica seja atendida. O objetivo foi melhorar a aplicação, permitindo que ela não terminasse após uma única operação, mas continuasse executando até que o usuário decida sair.

Principais Conceitos

1. O Loop "for"

- **Uso do `for` em Go:** Ao contrário de outras linguagens, Go utiliza apenas o loop `for` para repetição de código, tornando-o flexível.

Estrutura do `for`:

```go
for i := 0; i < 2; i++ {
    // Código a ser repetido
}
```

✔ **Inicialização (i := 0):** A variável `i` é iniciada com 0. ✔ **Condição (i < 2):** O loop continuará enquanto `i` for menor que 2. ✔ **Incremento (i++):** Após cada iteração, `i` será incrementado em 1.

Esse loop foi usado para executar as operações do programa várias vezes.

---

2. Código Dentro do Loop

- O professor sugeriu mover o bloco de código de escolha de ação (como depositar ou sacar) para dentro do loop `for`, o que fez a aplicação executar várias operações em sequência.

Exemplo:

```go
for {
    // Solicitação de escolha
    // Código de depósito, saque e verificação de saldo
}
```

✔ **Resultado:** O programa não para após uma operação, mas continua executando até o usuário optar por sair.

Código Atualizado com Loop `for`

```go
package main

import (
    "fmt"
)

func main() {
    var accountBalance float64 = 1000
    var choice int

    for {
        fmt.Println("Escolha uma opção:")
        fmt.Println("1 - Ver saldo")
        fmt.Println("2 - Depositar dinheiro")
        fmt.Println("3 - Sacar dinheiro")
        fmt.Println("4 - Sair")

        fmt.Print("Sua escolha: ")
        fmt.Scan(&choice)

        if choice == 1 {
            fmt.Println("Seu saldo atual é:", accountBalance)
        } else if choice == 2 {
```

```
            var depositAmount float64
            fmt.Print("Valor do depósito: ")
            fmt.Scan(&depositAmount)

            if depositAmount <= 0 {
                fmt.Println("Valor inválido! O depósito deve ser maior que zero.")
                return
            }

            accountBalance += depositAmount
            fmt.Println("Saldo atualizado! Novo saldo:", accountBalance)
        } else if choice == 3 {
            var withdrawalAmount float64
            fmt.Print("Valor do saque: ")
            fmt.Scan(&withdrawalAmount)

            if withdrawalAmount <= 0 {
                fmt.Println("Valor inválido! O saque deve ser maior que zero.")
                return
            }
            if withdrawalAmount > accountBalance {
                fmt.Println("Valor inválido! Você não pode sacar mais do que tem na
conta.")
                return
            }

            accountBalance -= withdrawalAmount
            fmt.Println("Saldo atualizado! Novo saldo:", accountBalance)
        } else {
            fmt.Println("Goodbye!")
            break
        }
    }
}
```

Explicação do Código

- **Loop `for`**: A principal mudança foi o uso do loop `for`, que mantém o programa rodando até que o usuário escolha a opção de sair.

- **Condição** `break`: Se o usuário escolher a opção 4 (sair), o comando `break` encerra o loop e o programa finaliza.
- **Continuação após cada operação**: Após cada ação (ver saldo, depositar, sacar), o programa exibe o menu novamente.

Pontos Importantes

✔ **Uso do loop** `for`: A única estrutura de repetição em Go, que pode ser usada de maneira flexível.
✔ **Controle do fluxo**: O programa continua executando até que o usuário opte por sair, ao contrário do comportamento anterior de término imediato.
✔ **Estrutura do loop infinito**: Usando `for {}` podemos criar um loop que repete indefinidamente até que uma condição de parada (como `break`) seja atendida.

Próximos Passos

🔹 **Melhorias no loop**: No próximo passo, o professor irá demonstrar como criar um loop infinito mais eficiente para que o programa continue executando até que o usuário decida sair explicitamente.
🔹 **Refinamento da interface**: Tornar o fluxo do programa mais intuitivo e robusto.

Se precisar de mais detalhes sobre o código ou qualquer conceito, estou à disposição!