Introdução ao Go

Aula 34: Loops Infinitos, "break" & "continue"

Nesta aula, o foco foi em como criar loops infinitos em Go e utilizar os comandos `break` e `continue` para controlar o fluxo dentro dos loops.

Principais Conceitos

1. Loops Infinitos

- Em Go, um loop infinito pode ser criado simplesmente usando o `for` sem uma condição. O loop continuará executando indefinidamente até que uma instrução de saída seja dada.

Exemplo de loop infinito:

```go
for {
    // Código a ser executado indefinidamente
}
```

Esse loop manterá a execução da aplicação, permitindo ao usuário continuar interagindo com ela até que uma condição para saída seja atingida.

---

2. Usando "return" para sair de uma função

- Inicialmente, para sair do loop e da aplicação, o professor usou a instrução `return`. Quando o usuário escolhia a opção de sair (número 4), a função `main()` era encerrada, fazendo com que o programa terminasse.

Exemplo de uso do `return`:

```go
```

```
if choice == 4 {
    fmt.Println("Goodbye!")
    return
}
```

Isso fez com que o programa fechasse imediatamente quando a opção de sair fosse escolhida.

---

3. Usando "break" para sair do loop

- O comando `break` é utilizado para interromper a execução de um loop, fazendo com que o código fora do loop seja executado em seguida. É uma alternativa ao uso de `return` quando queremos sair apenas do loop e não da função inteira.

Exemplo de uso do `break`:

go

```
if choice == 4 {
    fmt.Println("Goodbye!")
    break
}
```

Com isso, o programa exibe a mensagem "Goodbye!" e sai do loop, permitindo que o código fora do loop (como um agradecimento ao usuário) seja executado.

---

4. Usando "continue" para pular a iteração atual

- O comando `continue` faz com que a execução pule para a próxima iteração do loop, ignorando o restante do código dentro da iteração atual. Isso é útil quando queremos dar outra chance ao usuário sem sair do loop.

Exemplo de uso do `continue`:

- Se o usuário tentar realizar uma operação inválida (como depositar um valor negativo), o `continue` faz com que o loop seja retomado imediatamente, permitindo que o usuário tente novamente sem sair do programa.

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

Código Atualizado com "break" e "continue"

Aqui está um exemplo que usa tanto o `break` quanto o `continue` para controlar a execução do programa:

```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.")
                continue // Volta para o início do loop e pede novamente a escolha
            }

            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.")
                continue // Volta para o início do loop e pede novamente a escolha
            }
            if withdrawalAmount > accountBalance {
                fmt.Println("Valor inválido! Você não pode sacar mais do que tem na
conta.")
                continue // Volta para o início do loop e pede novamente a escolha
            }

            accountBalance -= withdrawalAmount
            fmt.Println("Saldo atualizado! Novo saldo:", accountBalance)
        } else if choice == 4 {
            fmt.Println("Obrigado por escolher nosso banco!")
            break // Sai do loop e do programa
        }
    }
}
```

Explicação do Código

- **Loop Infinito (`for {}`):** A aplicação continua executando até que o usuário escolha a opção de sair.
- **Comando `continue`:** Usado para pular a execução do código de saque ou depósito inválido e permitir que o usuário tente novamente sem sair do programa.
- **Comando `break`:** Interrompe o loop quando o usuário escolhe sair (opção 4), permitindo que o código após o loop (mensagem de agradecimento) seja executado.

Pontos Importantes

✔ **Loop Infinito (`for {}`):** Permite manter o programa executando até que uma condição de saída seja atendida. ✔ **Comando `break`:** Usado para sair do loop e continuar a execução do código após ele. ✔ **Comando `continue`:** Permite pular a execução do restante do código na iteração atual do loop e iniciar a próxima iteração.

Próximos Passos

🔹 **Refinamento do Loop:** Agora que o loop está funcionando bem com `break` e `continue`, podemos melhorar ainda mais a lógica de interações do usuário, como adicionar validações ou funcionalidades extras.
🔹 **Outras Considerações:** Explorar mais sobre controle de fluxo e otimização da interação do usuário.

Se precisar de mais detalhes ou quiser praticar algum exemplo, só avisar!