Introdução ao Go

Aula 23: Entendendo Funções

Nesta aula, exploramos o conceito de **funções** em Go, um elemento essencial para estruturar, reutilizar e organizar o código de maneira eficiente. Vamos entender o que são funções, como defini-las e usá-las, e como criar nossas próprias funções personalizadas.

O Que São Funções?

- Uma **função** é um bloco de código reutilizável que realiza uma tarefa específica.
- As funções só são executadas quando chamadas.
- Exemplos de funções que já utilizamos:
  - `fmt.Println()` e `fmt.Scan()`, ambas fornecidas pela biblioteca padrão de Go.
  - A função especial `main()`, que é o ponto de entrada do programa e chamada automaticamente pelo Go ao iniciar.

Por Que Usar Funções?

1. **Reutilização de Código**:
   - Reduz a duplicação, facilitando manutenção e ajustes.
2. **Organização**:
   - Divide o programa em blocos lógicos menores e mais compreensíveis.
3. **Flexibilidade**:
   - Recebe entradas dinâmicas (parâmetros) e pode retornar valores úteis.

Como Criar uma Função em Go?

1. Use a palavra-chave `func`.
2. Dê um nome à função.
3. Adicione **parênteses** para definir parâmetros (se houver).
4. Use **chaves** `{}` para encapsular o corpo da função.

Exemplo de Função Simples

Código

go

```go
package main

import (
    "fmt"
)

// Definição da função
func sayHello() {
    fmt.Println("Hello, World!")
}

func main() {
    // Chamando a função
    sayHello()
}
```

Saída

plaintext

```
Hello, World!
```

Adicionando Parâmetros

- **Parâmetros** são valores que podem ser passados para a função no momento da chamada.
- Sintaxe:
  - Dentro dos parênteses da função, declare:
    - Nome do parâmetro.
    - Tipo do parâmetro.

Código com Parâmetros

go

```go
package main

import (
    "fmt"
)

// Função com parâmetro
func greetUser(name string) {
    fmt.Printf("Hello, %s!\n", name)
}

func main() {
    // Chamando a função com um argumento
    greetUser("Alice")
    greetUser("Bob")
}
```

Saída

plaintext

```
Hello, Alice!
Hello, Bob!
```

Retornando Valores

- As funções podem **retornar valores** para o ponto onde foram chamadas.
- Para isso, declare o **tipo de retorno** após os parênteses na definição da função.
- Use a palavra-chave `return` para especificar o valor a ser retornado.

Código com Retorno

go

```go
package main

import (
    "fmt"
)

// Função que retorna o quadrado de um número
func square(number int) int {
    return number * number
}

func main() {
    // Armazenando o valor retornado
    result := square(5)
    fmt.Println("Square of 5 is:", result)
}
```

Saída

plaintext

```
Square of 5 is: 25
```

Funções com Múltiplos Parâmetros e Retornos

1. **Vários Parâmetros**:
    - Separe os parâmetros por vírgulas.
2. **Vários Retornos**:

- Separe os tipos de retorno com vírgulas.

Código

```go
package main

import (
    "fmt"
)

// Função que soma e multiplica dois números
func sumAndMultiply(a int, b int) (int, int) {
    sum := a + b
    product := a * b
    return sum, product
}

func main() {
    // Capturando múltiplos retornos
    sum, product := sumAndMultiply(3, 4)
    fmt.Println("Sum:", sum)
    fmt.Println("Product:", product)
}
```

Saída

```plaintext
Sum: 7
Product: 12
```

Funções Personalizadas: Um Exemplo

Podemos criar funções utilitárias para evitar repetição e organizar o código.

Código de Exemplo

```go
```

```go
package main

import (
    "fmt"
)

// Função para exibir uma mensagem formatada
func outputText(text string) {
    fmt.Println(text)
}

func main() {
    // Usando a função personalizada
    outputText("Bem-vindo ao programa!")
    outputText("Aprenda a criar funções em Go!")
}
```

Saída

```plaintext
Bem-vindo ao programa!
Aprenda a criar funções em Go!
```

Pontos-Chave Sobre Funções

1. **Parâmetros e Tipos**:
    - Cada parâmetro deve ter seu tipo especificado.
    - Parâmetros do mesmo tipo podem compartilhar o mesmo tipo na declaração:
        - Exemplo: `func example(a, b int)`.
2. **Valores de Retorno**:
    - Especifique o tipo de retorno após os parênteses:
        - Exemplo: `func example() int`.
3. **Erro em Funções**:

- Se uma função exige parâmetros e não recebe nenhum, o compilador apresentará erros.
- Exemplo:

```go
outputText() // Erro: parâmetro ausente
```

Resumo

- As funções são **blocos reutilizáveis de código** que organizam e otimizam os programas.
- Podem aceitar **parâmetros** e retornar valores para aumentar a flexibilidade.
- Criar e usar funções é essencial para construir programas robustos e modulares.

Seja com funções simples ou complexas, o uso delas é um dos pilares para dominar a programação em Go!