Sessão - Structs, Ponteiros e Composição em Go

39. Ponteiros:

Um ponteiro armazena o endereço de memória de outra variável.

Exemplo:

```
package main

import "fmt"

func alterarValor(num *int) {
    *num = 20
}

func main() {
    x := 10
    fmt.Println("Antes de alterar:", x)

    alterarValor(&x)
    fmt.Println("Depois de alterar:", x)
}
```

40. Go Mod:

O comando `go mod` gerencia dependências em Go.

Exemplo:

go mod init nome_do_projeto

41. Struct:

Uma struct é uma coleção de campos.

Exemplo:

```
package main

import "fmt"

type Pessoa struct {
    nome string
    idade int
}

func main() {
    p := Pessoa{nome: "Lincoln", idade: 25}
    fmt.Println("Nome:", p.nome)
    fmt.Println("Idade:", p.idade)
}
```

42. Encapsulamento:

Go implementa encapsulamento através de visibilidade pública e privada.

Exemplo:

```
package main

import "fmt"

type Pessoa struct {
    Nome  string
    idade int
}

func main() {
    p := Pessoa{Nome: "Lincoln", idade: 25}
    fmt.Println("Nome:", p.Nome)
}
```

43. Composição:

Go usa composição para incluir structs dentro de structs.

Exemplo:

```
package main

import "fmt"
```

```
type Endereco struct {
    Rua string
}

type Pessoa struct {
    Nome string
    Endereco
}

func main() {
    p := Pessoa{Nome: "Lincoln", Endereco: Endereco{Rua: "Rua A"}}
    fmt.Println("Nome:", p.Nome)
    fmt.Println("Rua:", p.Rua)
}
```

44. Métodos:

Métodos são funções associadas a structs.

Exemplo:

```
package main

import "fmt"

type Pessoa struct {
    Nome string
```

```
}

func (p Pessoa) Saudacao() {
    fmt.Println("Olá,", p.Nome)
}

func main() {
    p := Pessoa{Nome: "Lincoln"}
    p.Saudacao()
}
```

45. Alterando Dados por Métodos:

Métodos podem alterar valores usando ponteiros.

Exemplo:

```
package main

import "fmt"

type Pessoa struct {
    Nome string
}

func (p *Pessoa) AlterarNome(novoNome string) {
    p.Nome = novoNome
```

```
}

func main() {
    p := Pessoa{Nome: "Lincoln"}
    p.AlterarNome("José")
    fmt.Println("Nome alterado:", p.Nome)
}
```

46. Herança (Composição):

Go usa composição para reutilizar comportamento como herança.

Exemplo:

```
package main

import "fmt"

type Animal struct {
    Nome string
}

func (a Animal) Andar() {
    fmt.Println(a.Nome, "está andando.")
}

type Cachorro struct {
```

```
    Animal
}

func main() {
    c := Cachorro{Animal: Animal{Nome: "Rex"}}
    c.Andar()
}
```