



FUNÇÃO E PONTEIRO

FUNÇÕES



O QUE SÃO FUNÇÕES?

A ideia básica de uma função, implementada em alguma linguagem de programação, é encapsular um código que poderá ser invocado/chamado por qualquer outro trecho do programa.

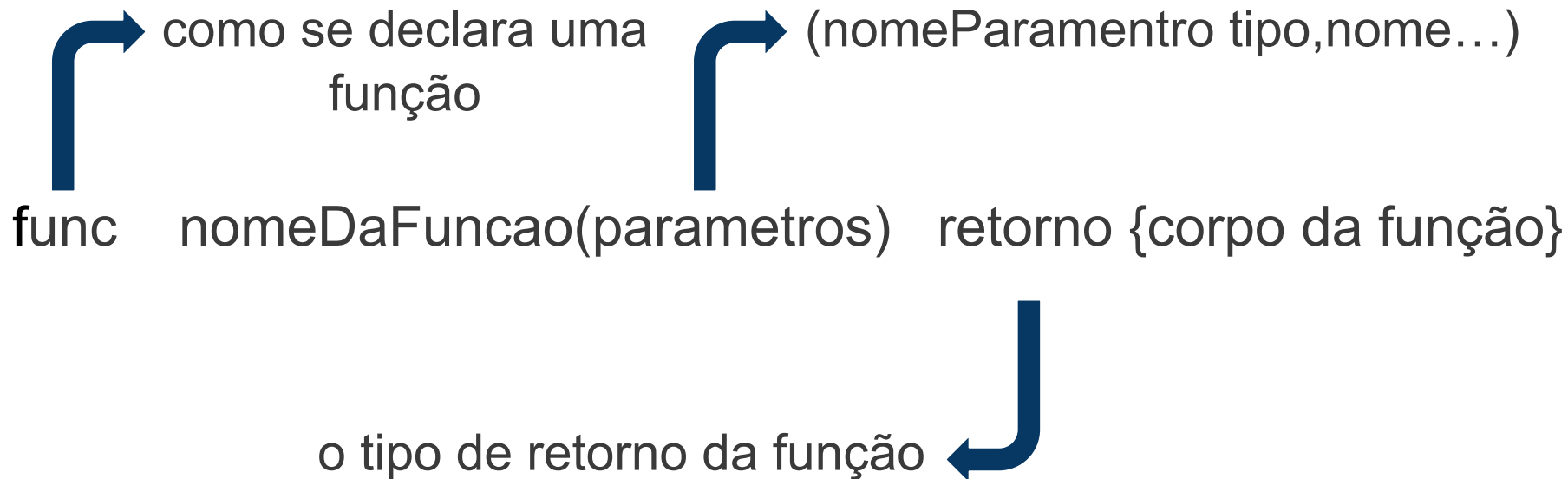


EXEMPLO

Preciso verificar se um número é par várias vezes durante meu programa, invés de copiar e colar um pedaço de código várias vezes, posso criar uma função

```
func descobrirPar(numero int) bool {  
    if numero%2 == 0 {  
        return true  
    }  
    return false  
}
```

COMPONENTES DA FUNÇÃO NO GO


func nomeDaFuncao(parametros) retorno {corpo da função}
como se declara uma função (nomeParametro tipo,nome...)
o tipo de retorno da função

```
func adicionar(x int, y int) {  
    total := 0  
    total = x + y  
    fmt.Println(total)  
}
```

DEFER

Defer trata-se de uma instrução para que uma função seja executada por último, independente da ordem

O defer é muito utilizado para realizar operações de limpeza, como por exemplo, fechar conexões com banco de dados, fechar stream de arquivos entre outros



```
func main() {  
    defer fmt.Println("Esse é o primeiro")  
    fmt.Println("Esse é o segundo")  
}
```

Esse é o segundo

Esse é o primeiro

Program exited.

MÉTODOS

Um método é uma função anexada a um tipo, permitindo que seja possível comunicar como os dados serão usados. Um método não está disponível para qualquer tipo de dado, deve ser utilizado apenas pelo tipo definido.



COMPONENTES DO MÉTODO NO GO



é aqui que o método se difere da função,
é declarado o tipo que o método vai ser
anexado

```
func (nomeTipo tipo) nomeDoMetodo(parametros) retorno {corpo do método}
```

```
type pessoa struct {  
    nome          string  
    sobrenome     string  
}  
  
func (p pessoa) bomDia() {  
    fmt.Println(p.nome, "diz bom dia")  
}  
  
func main() {  
    ana := pessoa{"Ana", "Silva"}  
    ana.bomDia()  
}
```

Ana diz bom dia

Program exited.

FUNÇÃO COMO EXPRESSÃO

Uma variável recebe como valor uma
função



```
func main() {  
    x := 5  
    y := func(x int){  
        fmt.Println(x,"vezes 5 é : ")  
        fmt.Println(x * 5 )  
    }  
    y(x)  
}
```

5 vezes 5 é :
25

Program exited.

RECURSIVIDADE

A recursividade consiste em uma função que pode chamar a si mesma, geralmente até atender alguma condição determinada




```
func main() {  
    fmt.Println(fatorial(4))  
}
```

```
func fatorial(x int ) int {  
    if x == 1 {  
        return x  
    }  
    return x * fatorial(x- 1)  
}
```

PONTEIRO



O QUE SÃO PONTEIROS?

Ponteiros são variáveis que apontam para um endereço de memória

São declarados usando *



```
func main(){  
    var criatura string = "tubarão"  
    var ponteiro *string = &criatura  
  
    fmt.Println("criatura = ", criatura)  
    fmt.Println("ponteiro = ", ponteiro)  
}
```

criatura = tubarão
ponteiro = 0xc00009a050

Program exited.

COMO IMPRIMIR O VALOR DE UM PONTEIRO?

Para imprimir o valor da variável
apontada é necessário usar

*



```
func main(){  
    var criatura string = "tubarão"  
    var ponteiro *string = &criatura  
  
    fmt.Println("criatura = ", criatura)  
    fmt.Println("ponteiro = ", ponteiro)  
    fmt.Println("*ponteiro = ", *ponteiro)  
}
```

```
criatura = tubarão  
ponteiro = 0xc00009a050  
*ponteiro = tubarão
```

Program exited.

MAS PARA QUE SERVE O PONTEIRO?

Em Go o ponteiro é muito utilizado para passar argumentos para função.



```
type Criatura struct {  
    Especie string  
}
```

```
func main() {  
    var criatura Criatura = Criatura{Especie: "tubarão"}  
  
    fmt.Printf("1) %+v\n", criatura)  
    changeCriatura(criatura)  
    fmt.Printf("3) %+v\n", criatura)  
}
```

```
func changeCriatura(criatura Criatura) {  
    criatura.Especie = "peixe"  
    fmt.Printf("2) %+v\n", criatura)  
}
```


1){Especie : tubarão}
2){Especie : peixe}
3){Especie : tubarão}

PONTEIRO COMO ARGUMENTO DE FUNÇÃO

Passar por referência, ou seja, passar um ponteiro para aquele argumento.

Dessa forma o valor da variável é alterado em todo o programa e não apenas no escopo da função



```
type Criatura struct {  
    Especie string  
}
```

```
func main() {  
    var criatura Criatura = Criatura{Especie: "tubarão"}  
  
    fmt.Printf("1) %+v\n", criatura)  
    changeCriatura(&criatura)  
    fmt.Printf("3) %+v\n", criatura)  
}
```

```
func changeCriatura(criatura *Criatura) {  
    criatura.Especie = "peixe"  
    fmt.Printf("2) %+v\n", criatura)  
}
```

1){Especie : tubarão}
2)&{Especie : peixe}
3){Especie : tubarão}

PARA USAR A CABEÇA

Faça uma função/método que receba uma string como parâmetro e que retorne uma nova string, onde a sequência dos caracteres foi invertida. Dentro da parte principal (main), leia uma string digitada pelo usuário e passe para a função/método criada, imprimindo em seguida a string devolvida.



```
import "fmt"
import "bufio"
import "os"

func main() {

    txt := bufio.NewReader(os.Stdin)
    fmt.Print("Digite o texto: ")
    text, _ := txt.ReadString('\n')

    inverted := invert(text)

    fmt.Println("Seu texto invertido: "+inverted)
}

func invert(text string) string {

    txt := []rune(text)
    ret := []rune{}

    /* O -2 é porque começa em 0, e para não pegar a quebra de linha */
    for i := len(txt) - 2; i >= 0; i-- {
        ret = append(ret, txt[i])
    }

    return string(ret)
}
```