Funções em Kotlin

MOV786205 – CST em Análise e Desenvolvimento de Sistemas

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Campus São José

Licenciamento

O que você vai aprender?

Funções Kotlin

- Parâmetros e retorno de funções
- Funções de ordem superior
- Funções anônimas
- Funções de extensão

Funções

- Definidas usando a palavra-chave `fun`, seguida pelo nome da função, parâmetros entre parênteses, tipo de retorno e o corpo da função entre chaves

```
package ads.mov.funcoes

fun saudacao(nome: String): String {
    return "Olá, $nome!"
}

fun main() {
    val resultado = saudacao("Juca")
    println(resultado)
}
```

Tipo de retorno

- Se uma função não retorna um valor, o tipo de retorno pode ser omitido ou declarado como `Unit`, que é o equivalente a `void` em outras linguagens

```
package ads.mov.funcoes

// Exemplo de função que não retorna valor
fun imprimirMensagem(mensagem: String) {
    println(mensagem)
}

fun naoRetornaNadaComUnit(): Unit {
    println("Esta função não retorna nada")
}

fun main() {
    imprimirMensagem("Olá, mundo!")
    naoRetornaNadaComUnit()
}
```

Parâmetros

- Parâmetros podem ter valores padrão, permitindo que a função seja chamada com menos argumentos
- Parâmetros nomeados permitem especificar quais argumentos estão sendo passados, melhorando a legibilidade

```
package ads.mov.funcoes

fun saudacaoPersonalizada(nome: String = "Visitante", saudacao: String = "Olá"): String {
    return "$saudacao, $nome!"
}

fun main() {
    println(saudacaoPersonalizada()) // Usa valores padrão
    println(saudacaoPersonalizada("Juca")) // Passando apenas o nome
    println(saudacaoPersonalizada(saudacao = "Bom dia")) // Passando apenas a saudação
    // Parâmetros nomeados
    println(saudacaoPersonalizada(saudacao = "Bom dia", nome = "Juca"))
}
```

Convenção de estilo

Para declaração de parâmetros e passagem de argumentos

- Se a função tiver muitos parâmetros, passe um argumento por linha para melhorar a legibilidade

```
package ads.mov.funcoes

fun calcularPreco(
    precoBase: Double,
    desconto: Double = 0.0,
    taxaServico: Double = 0.0
): Double {
    return precoBase - desconto + taxaServico
}
fun main() {
    val preco = calcularPreco(
        precoBase = 100.0,
        desconto = 10.0,
        taxaServico = 5.0
    )
    println("Preço: $preco")
}
```

Funções como expressões

- Quando o corpo da função é uma única expressão, você pode usar a sintaxe de expressão de função, que permite omitir a palavra-chave `return` e o tipo de retorno

```
package ads.mov.funcoes

fun saudacao(nome: String) = "Olá, $nome!"

fun main() {
    val resultado = saudacao("Juca")
    println(resultado)
}
```

Funções como cidadãos de primeira classe

Funções como cidadãos de primeira classe

- Funções são tratadas como **cidadãos de primeira classe**[1], o que significa que podem ser:
  - atribuídas a variáveis;
  - passadas como argumentos para outras funções; e
  - retornadas de outras funções
- Isso permite criar **funções de ordem superior**[2], que recebem outras funções como parâmetros ou retornam funções

[1] https://pt.wikipedia.org/wiki/Cidad%C3%A3o_de_primeira_classe
[2] https://en.wikipedia.org/wiki/Higher-order_function

Declaração de variáveis para armazenar funções

- O tipo de uma função é definido como:
  - `(parametros) -> tipoDeRetorno`

```
package ads.mov.funcoes

fun saudacao(nome: String, idade: Int): String {
    return "Olá, $nome! Você tem $idade anos."
}

fun main() {
    // Declaração de variável de função
    val aux: (String, Int) -> String

    // Atribuição da função à variável com o operador de chamada de função ::
    aux = ::saudacao

    // Atribuição da função à variável inferindo o tipo
    val auxInferido = ::saudacao

    // Chamada da função através da variável
    println(aux("Juca", 30))
}
```

Parâmetros e tipos de retorno

- Parâmetros podem ser omitidos com `()`, e o tipo de retorno pode ser `Unit` ou omitido

```
// Representa uma função que não recebe parâmetros e não retorna nada
val funcaoSemParametros: () -> Unit

// Representa uma função que não recebe parâmetros e retorna uma String
val funcaoSemParametrosERetorno: () -> String

// Representa uma função que recebe dois Int e retorna um Int
val funcaoComDoisParametros: (Int, Int) -> Int
```

Funções anônimas ou lambdas

- **Funções anônimas ou lambdas são funções sem nome** que podem ser atribuídas a variáveis ou passadas como argumentos
- Podem ser usadas para criar funções de ordem superior, como funções que recebem outras funções como parâmetros

```
package ads.mov.funcoes

fun main() {
    // Função lambda atribuída a uma variável com tipo explícito
    val ola: (String, Int) -> String = { nome, idade ->
        "Olá, $nome! Você tem $idade anos."
    }
    // Função lambda atribuída a uma variável com tipo inferido
    val saudacao = { nome: String, idade: Int -> "Olá, $nome! Você tem $idade anos." }

    // Chamada das funções lambda
    println(saudacao("Juca", 30))
    println(ola("Juca", 30))
}
```

Funções lambda com um único parâmetro

- Para funções lambda com um único parâmetro, não é necessário especificar o nome do parâmetro e use `it` para referenciá-lo

```
package ads.mov.funcoes

fun main() {
    // Função lambda com um único parâmetro e especificando o nome do parâmetro
    val saudacao: (String) -> String = { nome -> "Olá, $nome!" }

    // Chamada da função lambda
    println(saudacao("Juca"))

    // Outra forma de escrever a mesma função sem especificar o nome do parâmetro
    // O parâmetro é referenciado implicitamente como 'it'
    val saudacao2: (String) -> String = { "Olá, $it!" }

    // Chamada da função lambda
    println(saudacao2("Maria"))
}
```

Funções como tipo de retorno de outra função

- Funções podem ser retornadas de outras funções, permitindo criar funções que geram outras funções dinamicamente
- Isso é útil para criar funções de ordem superior que podem ser configuradas com diferentes comportamentos

```
package ads.mov.funcoes

fun saudacaoPersonalizada(saudacao: String): (String) -> String {
    return { nome: String -> "$saudacao, $nome!" }
}

fun main() {
    // Cria uma função de saudação personalizada
    val saudacaoBomDia = saudacaoPersonalizada("Bom dia")
    val saudacaoBoaTarde = saudacaoPersonalizada("Boa tarde")

    // Chama as funções geradas
    println(saudacaoBomDia("Juca"))
    println(saudacaoBoaTarde("Maria"))
}
```

Funções de ordem superior

Funções de ordem superior

Funções que recebem outras funções como parâmetros ou retornam funções

- Permitem criar abstrações poderosas e reutilizáveis, facilitando a programação funcional
- Podem ser usadas para manipular coleções e aplicar transformações
  - Exemplo: uma função que recebe outra função como parâmetro e aplica essa função a cada elemento de uma lista
- Para definir uma função de ordem superior, você especifica o tipo da função como um parâmetro, seguindo a sintaxe `(TipoParametro) -> TipoRetorno`
- Não é necessário especificar os nomes dos parâmetros da função de ordem superior, apenas os tipos

```
// Sintaxe de função de ordem superior
fun nomeDaFuncao(parametro: Tipo, funcao: (TipoParametro) -> TipoRetorno): TipoRetorno
```

Exemplo de função de ordem superior

- A função `calcular` recebe dois números e uma operação, aplica a operação a esses números e retorna o resultado

```
package ads.mov.funcoes
fun calcular(a: Int, b: Int, operacao: (Int, Int) -> Int): Int {
    return operacao(a, b)
}

fun main() {
    // Função de adição
    val soma: (Int, Int) -> Int = { x, y -> x + y }
    // Função de subtração
    val subtracao: (Int, Int) -> Int = { x, y -> x - y }

    // Chamadas da função de ordem superior com diferentes operações
    println(calcular(5, 3, soma))
    println(calcular(5, 3, subtracao))
}
```

Literais de função

- Em linguagens de programação, um literal é um valor fixo escrito diretamente no código, sem o uso de variáveis ou expressões
  - Exemplos: 10, "Olá, mundo!", 3.14
- Kotlin permite usar literais de função, que são funções anônimas definidas diretamente no local onde são usadas
- Isso é útil para passar funções como argumentos sem precisar definir uma função separada
- Expressões lambda e funções anônimas são consideradas como literais de função

Literais de função

```
package ads.mov.funcoes
fun calcular(a: Int, b: Int, operacao: (Int, Int) -> Int): Int {
    return operacao(a, b)
}

fun main() {
    // Usando literais de função diretamente com parâmetros nomeados
    val resultado = calcular(a = 5, b = 3, operacao = { x, y -> x + y })

    // Usando literal de função sem parênteses extras
    val resultado2 = calcular(5, 3) { x, y -> x - y }
}
```

- Se o último parâmetro for uma função, o literal pode ser passado diretamente, sem parênteses extras

Aula baseada em

JETBRAINS. **Kotlin Programming Language**. 2025. Disponível em: https://kotlinlang.org/docs/. Acesso em: 9 jul. 2025.