Desenvolvimento para Dispositivos Móveis

Funções e Modularização de Código

Prof. Bruno Azevedo

UNIP – Universidade Paulista

O que é uma função?

- Uma **função** é um bloco de código que realiza uma tarefa específica.
- Em Kotlin, funções podem receber parâmetros, retornar valores e ser reutilizadas em diferentes partes do programa.
- A sintaxe básica é:

```
fun nomeDaFuncao(parametros): TipoDeRetorno {
    // corpo da função
}
```

- O uso de funções favorece a organização, manutenção e reutilização do código.

Entendendo Funções

- Reutilização de código: "Temos um cálculo que será usado múltiplas vezes..."
- Modularidade: "Separamos o código de forma lógica, deixando-o mais organizado..."
- Facilidade de manutenção: "Agora podemos mudar em apenas um lugar..."
- Legibilidade: "Eu entendo melhor o código do meu colega..."

Declaração de Funções

- A declaração de uma função em Kotlin segue a estrutura:

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

  - `fun` é a palavra-chave que define uma função.
  - `nome` é o identificador da função.
  - `(nome: String)` são os parâmetros (podem ser opcionais).
  - `: String` é o tipo de retorno.
  - O corpo da função fica entre chaves `{}`.

Chamada de Função

- Uma função é chamada (invocada) pelo seu nome:

```
fun saudacao(nome: String): String {
    return "Olá, $nome!"
}
val mensagem = saudacao("Ana")
println(mensagem)
```

- A função pode ser chamada de dentro do `main()` ou de outras funções.
- Se a função tiver retorno, é possível armazenar seu resultado em uma variável.

Retorno de Funções

- O `return` define o valor que a função devolve.
- O tipo de retorno deve ser declarado após os parênteses:

```
fun soma(a: Int, b: Int): Int {
    return a + b
}
```

Exemplos Simples de Funções

- Função que calcula o quadrado de um número :

```
fun quadrado(x: Int): Int {
    return x * x
}
```

- Função que exibe uma mensagem:

```
fun mostrarMensagem() {
    println("Bem-vindo ao app!")
}
```

- Exemplos de chamadas para estas funções:

```
val resultado = quadrado(5)
mostrarMensagem()
```

Função sem parâmetros

- Uma função pode ser definida sem nenhum parâmetro:

```
fun saudacaoSimples() {
    println("Olá, mundo!")
}
```

- Exemplo de chamada:

```
saudacaoSimples()
```

Função com parâmetros

- Funções podem receber valores ao serem chamadas:

```
fun saudacaoPersonalizada(nome: String) {
    println("Olá, $nome!")
}
```

- Chamada:

```
saudacaoPersonalizada("João")
```

Função com valores padrão para parâmetros (opcionais)

- É possível definir valores padrão para parâmetros:

```
fun saudacaoComPadrao(nome: String = "Visitante") {
    println("Bem-vindo, $nome!")
}
```

- Chamada com e sem argumento:

```
saudacaoComPadrao("Ana") // Exibe Ana
saudacaoComPadrao()      // Exibe Visitante
```

Função com valores padrão para parâmetros (opcionais)

- Outro exemplo:

```
fun reformatar(
    texto: String,
    normalizarCaixa: Boolean = true,
    primeiraLetraMaiuscula: Boolean = true,
    separadorPalavras: Char = ' ',
) { // Corpo da Função }
```

Função com retorno

- Funções podem retornar valores:

```
fun somar(a: Int, b: Int): Int {
    return a + b
}
```

- O retorno pode ser armazenado em uma variável:

```
val resultado = somar(3, 4)
println(resultado)
```

Função de expressão única

- Para funções simples, é possível usar sintaxe reduzida:

```
fun dobrar(x: Int) = x * 2
```

- O tipo de retorno é inferido automaticamente.
- Exemplo de uso:

```
val y = dobrar(5)
println(y)
```

Uso do operador de segurança (? e ?.)

- Evita exceções de ponteiro nulo:

```
fun mostrarTamanho(texto: String?) {
    println(texto?.length)
}
```

- Se `texto` for `null`, o método `length` não é chamado.
- Uso:

```
mostrarTamanho("Oi")
mostrarTamanho(null)
```

- Se permitirmos a chamada do método `length` para um valor nulo sem usar `?.`, ocorre um `NullPointerException`.
- Por isso, o uso do operador de segurança é fundamental para evitar falhas no app.

Uso do operador Elvis (?:)

- Utilizado para definir valor padrão se a expressão for `null`:

```
fun mostrarUsuario(nome: String?) {
    val resultado = nome ?: "Desconhecido"
    println("Usuario: $resultado")
}
```

- Exemplos:

```
mostrarUsuario("Carla")
mostrarUsuario(null)
```

- Saída:

```
Carla
Desconhecido
```

Função com assinatura explícita

- Como já vimos, é possível declarar explicitamente o tipo dos parâmetros e do retorno:

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

- Facilita a legibilidade e ajuda a evitar ambiguidades no código.
- Auxilia os desenvolvedores durante a manutenção.
- Recomenda-se usar sempre que possível.

Função de extensão

- Permitem adicionar novas funções a classes existentes, sem modificá-las:

```
fun String.reverter(): String {
    return this.reversed()
}
```

- Uso:

```
val palavra = "Kotlin"
println(palavra.reverter()) // Exibe "niltoK"
```

- Ou seja, é uma forma de estender o comportamento de uma classe.
- Voltaremos a esse assunto na aula de orientação a objetos.

Função Anônima

- Em Kotlin, é possível declarar uma função sem nome (função anônima) e atribuí-la a uma variável:

```
val saudacao = fun(nome: String): String {
    return "Olá, $nome!"
}
println(saudacao("João")) // Imprime: Olá, João!
```

O que são funções lambda?

- Uma **função lambda** é uma forma concisa de declarar uma função anônima.
- São blocos de código que podem ser:
  - Armazenados em variáveis.
  - Passados como argumentos para outras funções.
  - Retornados de funções.
- Estrutura básica de uma lambda:

```
{ parâmetro(s) -> corpo da função }
```

- Exemplo:

```
val saudacao = { nome: String -> "Olá, $nome!" } // Retorna "Olá, $nome!"
println(saudacao("Joana")) // Exibe Olá, Joana!
```

- Em Kotlin, lambdas fazem parte da programação funcional moderna.

Por que usar funções lambda?

- Código mais limpo e expressivo: evita necessidade de declarar funções nomeadas simples.
- Evita boilerplate: ideal para operações simples como filtros, transformações e ações rápidas.
- Substituem interfaces funcionais anônimas do Java, com muito menos código:

```
// Java:
botao.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) { ... }
});

// Kotlin:
botao.setOnClickListener { ... }
```

O uso do `it`: parâmetro implícito

- Quando a lambda tem apenas um parâmetro, o Kotlin permite omitir o nome da variável:

```
val dobro = { x: Int -> x * 2 } // Com o nome da variável
val dobro2: (Int) -> Int = { it * 2 } // usando "it"
```

- O identificador `it` é o nome implícito do único parâmetro.
- Isso torna o código mais conciso:

```
val lista = listOf(1, 2, 3)
val dobrados = lista.map { it * 2 } // map é uma função de transformação
println(dobrados)
```

- Se a lambda tiver mais de um parâmetro, `it` não é permitido – os nomes devem ser declarados.

Passando lambdas como argumentos

- Podemos passar uma função lambda como argumento de outra função:

```
fun executarOperacao(x: Int, operacao: (Int) -> Int) {
    println(operacao(x))
}
```

- Chamando a função:

```
executarOperacao(5) { it * 3 }
```

- A lambda `{ it * 3 }` é passada como argumento e aplicada ao valor `5`.
- O resultado é impresso: `15`.
- Isso permite criar funções dinâmicas e reutilizáveis com base em parâmetros.

Fluxo de chamada e retorno de funções

- O programa inicia pela função `main()`.
- Cada função chamada transfere o controle para seu corpo.
- Ao encontrar `return`, o controle volta para o ponto de chamada.
- O valor retornado (se houver) pode ser usado ou ignorado.
- Exemplo:

```
fun dobro(x: Int): Int {
    return x * 2
}
fun main() {
    val r = dobro(4)
    println(r)
}
```

Atividade Prática 3

1. Criar um aplicativo que verifica a aprovação:
   - Função que recebe duas notas como parâmetros e retorna se aprovado ou não.
   - Exibição de uma mensagem indicando se o aluno foi aprovado (média $>= 7$) ou reprovado (média < 7).
2. Ampliação: Adicionar novas funcionalidades ao aplicativo.
   - Alterar a função para poder receber dois valores, inclusive nulo.
   - Em caso de duas notas iguais possibilitar a passagem de apenas um parâmetro.
   - Caso a nota passada seja nula, considerar o valor 5.
3. Cada aluno deve produzir um relatório de 1 a 3 páginas contendo:
   - Resumo teórico: Explicação clara dos conceitos de função, parâmetros e valores de retorno.
   - Código-fonte comentado: Explicação detalhada do código desenvolvido, com foco nas funções criadas e sua chamada no aplicativo.