



Paradigma procedural em



Integrantes: André Gomes, Diogo Belshoff e Daniel Frigini

Introdução

Embora Kotlin seja conhecida por suas capacidades em programação orientada a objetos e funcional, o paradigma procedural desempenha um papel relevante, especialmente em cenários onde a simplicidade e a clareza na execução de tarefas sequenciais são essenciais.

Abordado

Características Principais
Exemplo de Funcionamento
A Linguagem Kotlin
Aplicações

O Paradigma Procedural

O que é?

modelo de programação baseado na execução sequencial de instruções e na organização do código em procedimentos ou funções.

foca em uma sequência de passos que devem ser seguidos para realizar uma tarefa específica.





Características do paradigma

- **Sequência de Instruções**
- **Procedimentos ou Funções**
- **Variáveis Globais e Locais**
- **Modularidade**
- **Estruturas de Controle**



Para evitar repetição, os códigos são encapsulados



Permite que o código seja dividido em partes menores e mais gerenciáveis.

Exemplo de Funcionamento

Vantagens

- Simplicidade
- Reutilização de Código
- Clareza



facilitando a manutenção e atualização do programa.



O fluxo de execução é geralmente linear

Exemplo de Funcionamento

Desvantagens

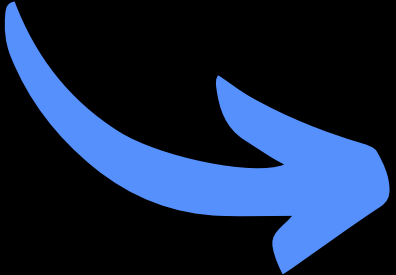
- Escalabilidade

pode se tornar difícil de gerenciar, com funções e variáveis globais

- Falta de Abstração

Não possui os mesmo níveis que outros paradigmas, como OO

A Linguagem Kotlin



**desenvolvida pela JetBrains e
lançada em 2011**



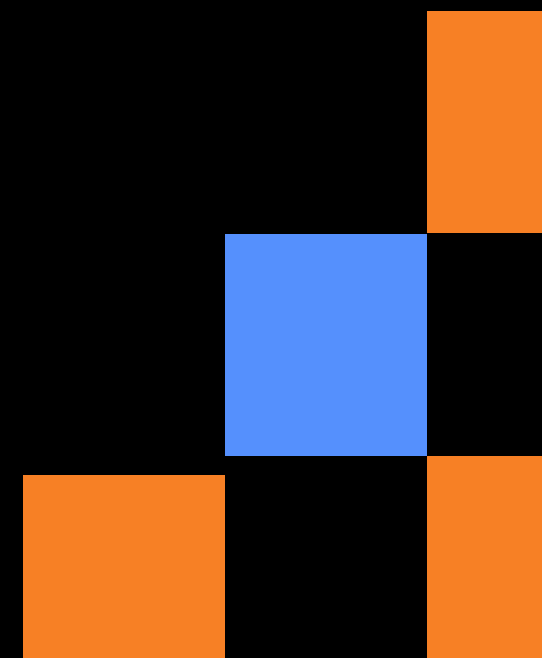
Criada para interoperar totalmente com Java



**linguagem oficial para o desenvolvimento de aplicativos
Android em 2017**



**mantém compatibilidade com a vasta base de
código Java**



Principais características da linguagem

Sintaxe Concisa e Expressiva

reduz a quantidade de código boilerplate necessário

Interoperabilidade com Java

permite a chamada de código Java a partir de Kotlin e vice-versa

Null Safety

sistema de tipos que previne erros de tempo de execução relacionados a valores nulos

Principais características da linguagem

Coroutines

simplificando o trabalho com código assíncrono e permitindo a execução de tarefas de forma não bloqueante

Programação Orientada a Objetos e Funcional

Permite uma abordagem híbrida dentro do mesmo projeto.

Compatibilidade Multiplataforma

suporte para Android, iOS, Web e servidor

Aplicações de Kotlin



Desenvolvimento Android



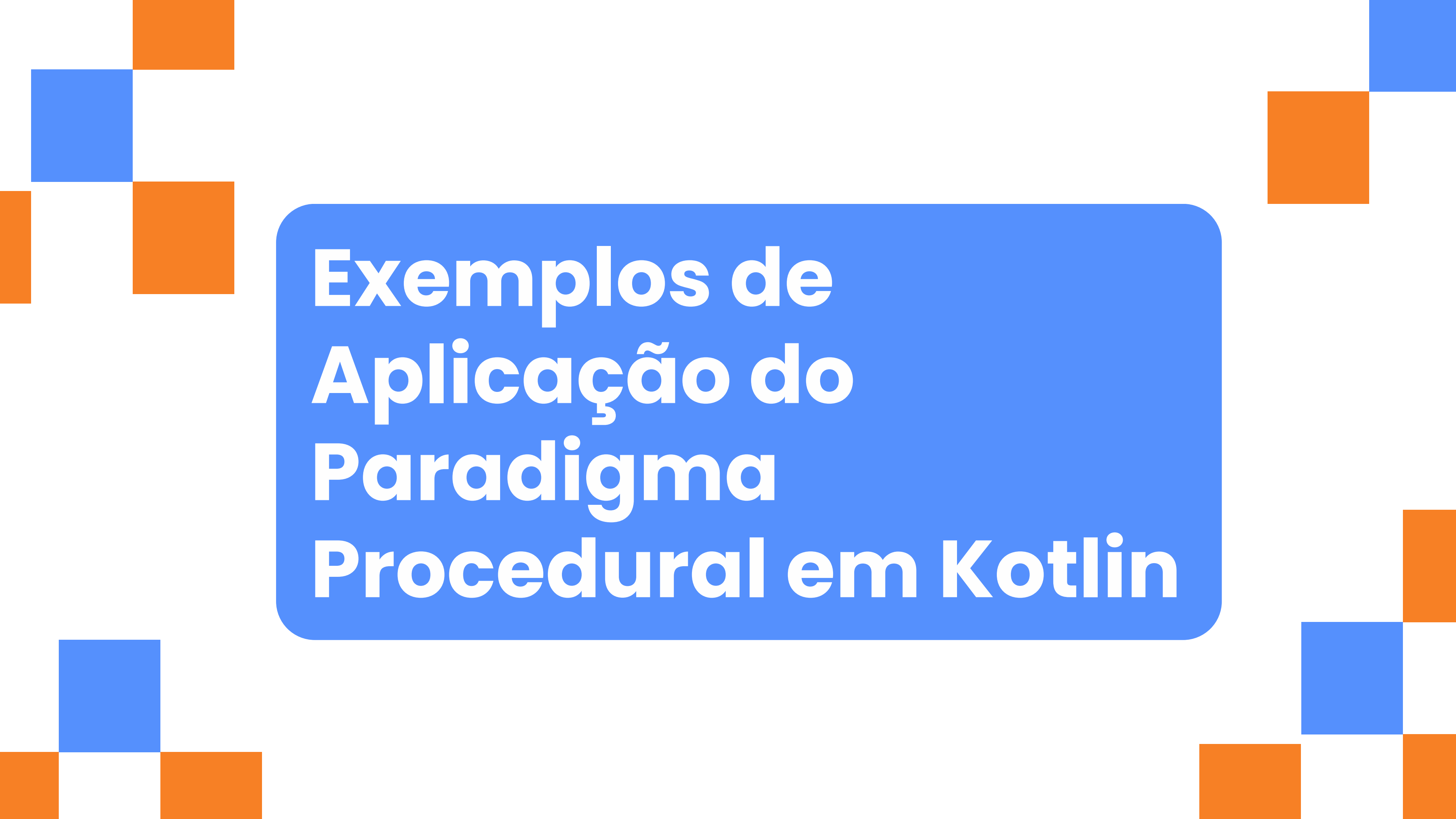
Aplicações Desktop e Web



Desenvolvimento Backend



Como frameworks como Ktor

The background features a pattern of orange and blue squares arranged in a grid-like fashion, with some squares missing, creating a decorative border around the central text box.

Exemplos de Aplicação do Paradigma Procedural em Kotlin

Cálculo da Soma de uma lista de números

```
fun sumList(numbers: List<Int>): Int {  
    var sum = 0  
    for (number in numbers) {  
        sum += number  
    }  
    return sum  
}  
  
fun main() {  
    val numbers = listOf(1, 2, 3, 4, 5)  
    val result = sumList(numbers)  
    println("A soma dos números é: $result")  
}
```

A função `sumList` percorre cada número da lista `numbers`, acumulando a soma em uma variável local `sum`. A função é então chamada no `main`, onde uma lista de inteiros é passada como argumento, e o resultado é impresso.

Encontrar o maior número em uma lista

```
fun findMax(numbers: List<Int>): Int {  
    var max = numbers[0]  
    for (number in numbers) {  
        if (number > max) {  
            max = number  
        }  
    }  
    return max  
}  
  
fun main() {  
    val numbers = listOf(3, 9, 2, 8, 6)  
    val maxNumber = findMax(numbers)  
    println("O maior número é: $maxNumber")  
}
```

A função `findMax` percorre a lista de números, comparando cada elemento com o atual maior valor armazenado em `max`. Se um número maior for encontrado, ele substitui o valor de `max`. O maior número é então retornado e impresso no `main`.

Verificação de Número Primo

```
fun isPrime(number: Int): Boolean {
    if (number < 2) return false
    for (i in 2 until number) {
        if (number % i == 0) {
            return false
        }
    }
    return true
}

fun main() {
    val num = 17
    if (isPrime(num)) {
        println("$num é um número primo")
    } else {
        println("$num não é um número primo")
    }
}
```

A função `isPrime` verifica se um número é divisível por qualquer número menor que ele, começando de 2. Se o número for divisível por algum desses números, ele não é primo. Caso contrário, é considerado primo. No `main`, a função é utilizada para verificar e imprimir se o número 17 é primo.

Fatorial de um Número

```
fun factorial(n: Int): Int {  
    var result = 1  
    for (i in 2..n) {  
        result *= i  
    }  
    return result  
}  
  
fun main() {  
    val num = 5  
    val fact = factorial(num)  
    println("O fatorial de $num é: $fact")  
}
```

A função **factorial** multiplica os números de 2 até o número **n**, armazenando o resultado em **result**. O fatorial é então retornado e exibido no **main**.

Verificação de Palíndromo

```
fun isPalindrome(word: String): Boolean {  
    val cleanedWord = word.replace("\\s".toRegex(), "").toLowerCase()  
    val reversedWord = cleanedWord.reversed()  
    return cleanedWord == reversedWord  
}
```

```
fun main() {  
    val word = "arara"  
    if (isPalindrome(word)) {  
        println("$word é um palíndromo")  
    } else {  
        println("$word não é um palíndromo")  
    }  
}
```

A função `isPalindrome` remove espaços e converte a string para minúsculas, depois verifica se a palavra é igual ao seu reverso. Se for, a função retorna `true`, indicando que a palavra é um palíndromo.